IBM® DB2 Universal Database™

Guía de desarrollo de aplicaciones: Programación de aplicaciones de cliente

Versión 8

SC10-3723-00

IBM® DB2 Universal Database™

Guía de desarrollo de aplicaciones: Programación de aplicaciones de cliente

Versión 8

SC10-3723-00

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

Este manual es la traducción del original inglés *IBM DB2 Universal Database Application Development Guide: Programming Client Applications Version 8* (SC09-4826-00).

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

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

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

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

Contenido

Parte 4. Otras interfaces de programación 361

Acerca de este manual

El manual *Guía de desarrollo de aplicaciones* es un libro de tres volúmenes que describe lo que tiene que saber sobre codificación, depuración, creación y ejecución de aplicaciones DB2:

- El manual *Guía de desarrollo de aplicaciones: Programación de aplicaciones de cliente* contiene lo que tiene que saber para codificar aplicaciones DB2 autónomas que se ejecutan en clientes DB2. Incluye información sobre:
  - Interfaces de programación que reciben soporte de DB2. Se proporcionan descripciones de alto nivel para DB2 Developer's Edition, interfaces de programación soportadas, recursos para crear aplicaciones Web y funciones de programación proporcionadas por DB2, como rutinas y activadores.
  - La estructura general que debe seguir una aplicación DB2. Se proporcionan recomendaciones sobre cómo mantener valores de datos y relaciones en la base de datos, se describen consideraciones sobre autorización y se proporciona información sobre cómo probar y depurar la aplicación.
  - SQL incorporado, tanto dinámico como estático. Se describen consideraciones generales sobre SQL incorporado, así como temas específicos que se aplican al uso de SQL estático y dinámico en aplicaciones DB2.
  - Lenguajes interpretados y de sistema principal, como C/C++, COBOL, Perl y REXX, y cómo utilizar SQL incorporado en aplicaciones escritas en estos lenguajes.
  - Java (tanto JDBC como SQLj) y consideraciones sobre la creación de aplicaciones Java que utilizan WebSphere Application Servers.
  - IBM OLE DB Provider para servidores DB2. Se proporciona información general sobre el soporte de IBM OLE DB Provider para servicios OLE DB, componentes y propiedades. También se proporciona información específica sobre aplicaciones Visual Basic y Visual C++ que utilizan la interfaz OLE DB para Objetos de datos ActiveX (ADO).
  - Temas relacionados con el soporte de idiomas nacionales. Se describen temas generales, como secuencias de clasificación, obtención de páginas de códigos y entornos locales y conversiones de caracteres. También se describen temas más específicos, como páginas de códigos DBCS, juegos de caracteres EUC y temas que se aplican a entornos EUC y UCS-2 de japonés y chino tradicional.

  - Gestión de transacciones. Se describen temas relacionados con aplicaciones que realizan actualizaciones múltiples y con aplicaciones que realizan transacciones simultáneas.
  - Aplicaciones en entornos de bases de datos particionadas. Se describen DSS dirigida, elusión local, inserciones colocadas en almacenamiento intermedio y resolución de aplicaciones en entornos de bases de datos particionadas.
  - Técnicas de aplicaciones utilizadas normalmente. Se proporciona información sobre cómo utilizar columnas generadas y de identidad y tablas temporales declaradas y sobre cómo utilizar puntos de rescate para gestionar transacciones.
  - Las sentencias de SQL que se pueden utilizar en aplicaciones de SQL incorporado.
  - Aplicaciones que acceden a entornos de sistema principal e iSeries. Se describen temas relacionados con aplicaciones de SQL incorporado que acceden a entornos de sistema principal e iSeries.
  - La simulación del orden binario EBCDIC.
- El manual *Guía de desarrollo de aplicaciones: Programación de aplicaciones de servidor* contiene lo que tiene que saber sobre programación en el servidor, que incluye rutinas, objetos grandes, tipos definidos por el usuario y activadores. Incluye información sobre:
  - Rutinas (procedimientos almacenados, funciones definidas por el usuario y métodos), que incluyen:
    - Rendimiento de rutinas, seguridad, consideraciones sobre la gestión de bibliotecas y restricciones.
    - Cómo registrar y escribir rutinas, incluidas las sentencias CREATE y depuración.
    - Modalidades de parámetros de procedimientos y manejo de parámetros.
    - Conjuntos de resultados de procedimientos.
    - Funciones de UDF que incluyen borradores y funciones escalares y de tabla.
    - Procedimientos de SQL que incluyen depuración y manejo de condiciones.
    - Estilos de parámetros, autorizaciones y vinculación de rutinas externas.
    - Consideraciones específicas de cada lenguaje para C, Java y rutinas de automatización OLE.
    - Invocación de rutinas
    - Selección de funciones y cómo pasar tipos diferenciados y LOB a funciones.

- Páginas de códigos y rutinas.

– Objetos grandes, que incluyen uso de LOB y localizadores, variables de referencia y datos CLOB.

– Tipos diferenciados definidos por el usuario, que incluyen tipificación estricta, definición y eliminación de UDT, creación de tablas con tipos estructurados, utilización de tipos diferenciados y tablas tipificadas para aplicaciones específicas, manipulación de tipos diferenciados y difusión entre los mismos y realización de comparaciones y asignaciones con tipos diferenciados, que incluyen operaciones UNION sobre columnas tipificadas de forma diferenciada.

– Tipos estructurados definidos por el usuario, que incluyen almacenamiento de instancias y creación de instancias, jerarquías de tipos estructurados, definición del comportamiento de los tipos estructurados, distribución dinámica de métodos, funciones de comparación, difusión y constructor y métodos mutador y observador correspondientes a tipos estructurados.

– Tablas tipificadas, que incluyen creación, eliminación, sustitución y almacenamiento de objetos, definición de identificadores de objetos generados por el sistema y restricciones en las columnas de identificador.

– Tipos de referencia, que incluyen relaciones entre objetos de tablas tipificadas, relaciones semánticas con referencias e integridad referencial frente a referencias de ámbito.

– Tablas y vistas tipificadas, que incluyen tipos estructurados como tipos de columnas, funciones de transformación y grupos de transformación, correlaciones de programas de lenguaje principal y variables del lenguaje principal de tipos estructurados.

– Activadores, que incluyen los activadores INSERT, UPDATE y DELETE, interacciones con restricciones referenciales, líneas generales de creación, granularidad, hora de activación, tablas y variables de transición, acciones desencadenadas, varios activadores y sinergia entre activadores, restricciones y rutinas.

• El manual *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones* contiene lo que tiene que saber para crear y ejecutar aplicaciones DB2 en los sistemas operativos a los que da soporte DB2:

– AIX

– HP-UX

– Linux

– Solaris

– Windows

Incluye información sobre:

- Cómo configurar el entorno de desarrollo de aplicaciones; incluye instrucciones específicas para procedimientos Java y SQL, cómo configurar la base de datos de ejemplo y cómo migrar las aplicaciones desde versiones anteriores de DB2.
- Servidores y software soportados por DB2 para crear aplicaciones, incluidos compiladores e intérpretes.
- Los archivos de los programas de ejemplo de DB2, makefiles, archivos de creación y archivos del programa de utilidad de comprobación de errores.
- Cómo crear y ejecutar applets, aplicaciones y rutinas Java.
- Cómo crear y ejecutar procedimientos de SQL.
- Cómo crear y ejecutar rutinas y aplicaciones C/C++.
- Cómo crear y ejecutar rutinas y aplicaciones COBOL de IBM y Micro Focus.
- Cómo crear y ejecutar aplicaciones REXX en AIX y Windows.
- Cómo crear y ejecutar con Objetos de datos ActiveX (ADO) mediante Visual Basic y Visual C++ en Windows.
- Cómo crear y ejecutar aplicaciones con objetos de datos remotos mediante Visual C++ en Windows.

Parte 1. Introducción

Capítulo 1. Visión general de las interfaces de programación soportadas

DB2 Developer's Edition

Las siguientes secciones describen DB2 Developer's Edition y dónde encontrar información sobre cómo instalar productos en el mismo.

Productos de DB2 Developer's Edition

DB2 Universal Database proporciona dos paquetes de productos para el desarrollo de aplicaciones: DB2 Personal Developer's Edition y DB2 Universal Developer's Edition. Personal Developer's Edition proporciona los productos DB2 Universal Database y DB2 Connect Personal Edition que se ejecutan en Linux y en sistemas operativos Windows. DB2 Universal Developer's Edition proporciona productos DB2 en estas plataformas, así como en AIX, en HP-UX y en entorno operativo Solaris. Póngase en contacto con el representante de IBM para ver una lista completa de plataformas soportadas.

Con el software que viene con estos productos puede desarrollar y probar aplicaciones que se ejecutan en un sistema operativo y que acceden a bases de datos en el mismo o en otro sistema operativo. Por ejemplo, puede crear una aplicación que se ejecute en el sistema operativo Windows NT pero que acceda a una base de datos en una plataforma UNIX como AIX. Consulte el Acuerdo de licencia para ver los términos y condiciones de uso de los productos Developer's Edition.

Personal Developer's Edition contiene varios CD-ROM con todo el código que necesita para desarrollar y probar sus aplicaciones. En cada caja, encontrará:

- Los CD-ROM del producto DB2 Universal Database para Linux y sistemas operativos Windows. Cada CD-ROM contiene el servidor DB2, Administration Client, Application Development Client y Run-Time Client para un sistema operativo soportado. Estos CD-ROM se proporcionan sólo para que pruebe sus aplicaciones. Si tiene que instalar y utilizar una base de datos, debe obtener una licencia válida adquiriendo el producto Universal Database.
- DB2 Connect Personal Edition
- Un CD-ROM de publicaciones de DB2 que contiene manuales de DB2 en formato PDF
- DB2 Extenders (sólo Windows)
- DB2 XML Extender (sólo Windows)
- VisualAge para Java, Entry Edition

Universal Developer's Edition contiene CD-ROM para todos los sistemas operativos que reciben soporte de DB2, e incluyen lo siguiente:

- DB2 Universal Database Personal Edition, Workgroup Server Edition y Enterprise Server Edition
- DB2 Connect Personal Edition y DB2 Connect Enterprise Edition
- Clientes de administración para todas las plataformas. Estos clientes contienen herramientas para administrar bases de datos, como el Centro de control y el Analizador de sucesos. Estos clientes también le permiten ejecutar aplicaciones en cualquier sistema.
- Clientes de desarrollo de aplicaciones para todas las plataformas. Estos clientes tienen herramientas de desarrollo de aplicaciones, programas de ejemplo y archivos de cabecera. Cada cliente DB2 AD incluye todo lo que necesita para desarrollar sus aplicaciones.
- Clientes de tiempo de ejecución para todas las plataformas. Una aplicación se puede ejecutar desde un cliente de tiempo de ejecución en cualquier sistema. El cliente de tiempo de ejecución no tiene algunas de las funciones del cliente de administración, como el Centro de control de DB2 y el Analizador de sucesos, de modo que ocupa menos espacio.
- DB2 Extenders
- DB2 XML Extender
- VisualAge para Java, Professional Edition (Windows)
- Websphere Studio
- Websphere Application Server, Standard Edition
- Recurso de gestión de consultas (probar y comprar)

Además, para ambas Developer's Editions, obtiene copias de otro software que puede encontrar útil para desarrollar aplicaciones. Este software puede variar de tanto en tanto y viene acompañado de acuerdos de licencia para uso.

Instrucciones para instalar productos DB2 Developer's Edition

Para obtener instrucciones sobre cómo instalar un producto disponible con DB2 Developer's Edition, consulte el manual *Guía rápida de iniciación* adecuado, disponible en el CD de PDF, o consulte el propio CD del producto para ver instrucciones de instalación.

Herramientas de DB2 Universal Database para desarrollar aplicaciones

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

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

Centro de control

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

DB2 también proporciona los siguientes recursos:

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

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

Diario Se utiliza para ver los siguientes tipos de información: toda la información disponible sobre trabajos cuya ejecución está pendiente,

que se están ejecutando o que se han terminado de ejecutar, el registro histórico de recuperación, el registro de alertas y el registro de mensajes. También puede utilizar el Diario para revisar los resultados de trabajos que se ejecutan de forma desatendida.

Centro de alertas
Se utiliza para supervisar el sistema en busca de avisos anteriores o de problemas potenciales o para automatizar acciones para corregir problemas.

Valor de herramientas
Se utiliza para cambiar los valores correspondientes al Centro de control, Centro de alertas y Duplicación.

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

Visual Explain

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

Interfaces de programación soportadas

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

Interfaces de programación que reciben soporte de DB2

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

- Utilizar las API de DB2 para realizar funciones administrativas como copia de seguridad y restauración de bases de datos.
- Incorporar sentencias de SQL estático y dinámico en sus aplicaciones.
- Codificar llamadas a funciones de Interfaz de nivel de llamada de DB2 (CLI de DB2) en sus aplicaciones para invocar sentencias de SQL dinámico.
- Desarrollar aplicaciones y applets Java que llaman a la interfaz de programación de aplicaciones Java Database Connectivity (API JDBC).
- Desarrollar aplicaciones Microsoft Visual Basic y Visual C++ que cumplen con las especificaciones de Objeto de acceso a datos (DAO) y Objeto de

datos remotos (RDO) y aplicaciones Objeto de datos ActiveX (ADO) que utilizan el puente Object Linking and Embedding Database (OLE DB).
- Desarrollar aplicaciones mediante herramientas de IBM o de otros proveedores como Net.Data, Excel, Perl y herramientas de usuario final de Open Database Connectivity (ODBC) como Lotus Approach y su lenguaje de programación, LotusScript.

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

Aparte del modo en que la aplicación accede a los datos, también debe tener en cuenta lo siguiente:
- Control de valores de datos utilizando:
  - Tipos de datos (integrados o definidos por el usuario)
  - Restricciones de comprobación de tablas
  - Restricciones de integridad referencial
  - Vistas utilizando la opción CHECK
  - Lógica de aplicación y tipos de variable
- Control de las relaciones entre valores de datos utilizando:
  - Restricciones de integridad referencial
  - Activadores
  - Lógica de aplicación
- Ejecución de programas en el servidor utilizando:
  - Procedimientos almacenados
  - Funciones definidas por el usuario
  - Activadores

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

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

La principal ventaja de transferir lógica centrada en los datos de la aplicación a la base de datos es que la aplicación pasa a ser más independiente de los datos. La lógica asociada a los datos se centraliza en un lugar, la base de

datos. Esto significa que puede cambiar datos o lógica de datos una vez de forma que ello afecte a *todas* las aplicaciones inmediatamente.

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

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

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

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

Conceptos relacionados:

- “Interfaz de nivel de llamada de DB2 (CLI) frente a SQL dinámico incorporado” en la página 169
- “SQL incorporado” en la página 9
- “Interfaz de nivel de llamada de DB2” en la página 11
- “Interfaces de programación de aplicaciones de DB2” en la página 8
- “Objetos de datos ActiveX y Objetos de datos remotos” en la página 17
- “DBI Perl” en la página 18
- “Herramientas de usuario final de ODBC” en la página 18
- “Herramientas para crear aplicaciones Web” en la página 19
- “Java Database Connectivity (JDBC)” en la página 15

Interfaces de programación de aplicaciones de DB2

Puede que sus aplicaciones tengan que realizar algunas tareas de administración de bases de datos, como crear, activar, hacer copia de seguridad o restaurar una base de datos. DB2 proporciona numerosas API para que pueda realizar estar tareas desde sus aplicaciones, incluidas

aplicaciones de SQL incorporado y CLI de DB2. Esto le permite programar en sus aplicaciones las mismas funciones administrativas que puede realizar mediante las herramientas de administración del servidor de DB2 disponibles en el Centro de control.

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

Conceptos relacionados:

- "Consideraciones sobre autorización para API" en la página 63
- "API administrativas en SQL incorporado o en programas de CLI de DB2" en la página 52

SQL incorporado

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

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

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

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

Por el contrario, las sentencias de SQL dinámico son aquellas que la aplicación crea y ejecuta en el tiempo de ejecución. Una aplicación interactiva que

solicita al usuario final partes clave de una sentencia de SQL, como los nombres de las tablas y las columnas que hay que buscar, es un buen ejemplo de SQL dinámico. La aplicación crea la sentencia de SQL mientras se está ejecutando y luego somete la sentencia para que se procese.

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

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

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

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

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

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

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

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

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

Conceptos relacionados:

- "SQL incorporado en aplicaciones REXX" en la página 372
- "Sentencias de SQL incorporado en C y C++" en la página 182
- "Sentencias de SQL incorporado en COBOL" en la página 237
- "Sentencias de SQL incorporado en FORTRAN" en la página 267
- "Sentencias de SQL incorporado en Java" en la página 306
- "SQL incorporado para Java (SQLj)" en la página 17

Tareas relacionadas:

- "Incorporación de sentencias de SQL en un lenguaje principal" en la página 77

Interfaz de nivel de llamada de DB2

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

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

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

`http://www.ibm.com/software/data/db2/udb/staticcli`

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

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

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

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

Conceptos relacionados:

- “API administrativas en SQL incorporado o en programas de CLI de DB2” en la página 52
- “Interfaz de nivel de llamada de DB2 (CLI) frente a SQL dinámico incorporado” en la página 169
- “Ventajas de CLI de DB2 sobre SQL incorporado” en la página 171
- “Cuándo utilizar CLI de DB2 o SQL incorporado” en la página 173
- “CLI de DB2 frente a SQL dinámico incorporado” en la página 13

Consulta relacionada:

- “Programas de ejemplo de la CLI de DB2” en el manual *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

CLI de DB2 frente a SQL dinámico incorporado

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

Las ventajas de CLI de DB2 son las siguientes:

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

No hay vinculación
No tiene que vincular aplicaciones CLI de DB2 individuales a cada base de datos a la que acceden. Sólo tiene que vincular los archivos que se suministran con CLI de DB2 una vez para todas las aplicaciones CLI de DB2. Esto permite reducir significativamente la cantidad de tiempo empleada en gestionar las aplicaciones.

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

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

Conversión de datos ampliada
CLI de DB2 convierte automáticamente datos entre los tipos de datos SQL y C. Por ejemplo, al captar cualquier tipo de

datos SQL en un tipo de datos char de C se convierten en una representación de serie de caracteres. Esto hace que CLI de DB2 resulte ideal para aplicaciones de consulta interactiva.

No hay áreas de datos globales

CLI de DB2 elimina la necesidad de disponer de áreas de datos globales controladas por la aplicación y normalmente complejas, como SQLDA y SQLCA, que suelen estar asociadas con aplicaciones de SQL incorporado. En su lugar, CLI de DB2 asigna y control automáticamente las estructuras de datos necesarias y proporciona un descriptor de contexto para que la aplicación haga referencia a las mismas.

Recuperar conjuntos de resultados de procedimientos almacenados

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

Cursores desplazables

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

Las ventajas de SQL dinámico incorporado son las siguientes:

Seguridad granular

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

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

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

Conceptos relacionados:

- "Interfaz de nivel de llamada de DB2 (CLI) frente a SQL dinámico incorporado" en la página 169
- "Ventajas de CLI de DB2 sobre SQL incorporado" en la página 171
- "Cuándo utilizar CLI de DB2 o SQL incorporado" en la página 173

Java Database Connectivity (JDBC)

El soporte de Java de DB2 incluye JDBC, una interfaz de SQL dinámico que no depende del proveedor que proporciona acceso a datos a la aplicación a través de métodos Java estandarizados. JDBC se parece a CLI de DB2 en que el usuario no tiene que precompilar ni vincular un programa JDBC. Como estándar independiente del proveedor, las aplicaciones JDBC ofrecen mayor portabilidad. Una aplicación escrita utilizando JDBC sólo utiliza SQL dinámico.

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

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

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

JDBC de tipo 2

Las aplicaciones Java basadas en el controlador JDBC de tipo 2 confían en el cliente DB2 para establecer conexión con DB2. Puede iniciar la aplicación desde el escritorio o desde la línea de mandatos, como cualquier otra aplicación. El controlador JDBC de DB2 maneja las llamadas a la API JDBC desde la aplicación y utiliza la conexión del cliente para comunicar las peticiones al servidor y para recibir los resultados. No puede crear applets Java mediante el controlador JDBC de tipo 2.

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

JDBC de tipo 3

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

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

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

JDBC de tipo 4

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

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

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

Conceptos relacionados:

• “Comparación entre SQLj y JDBC” en la página 284

Tareas relacionadas:

- “Codificación de aplicaciones y applets JDBC” en la página 294

SQL incorporado para Java (SQLj)

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

Para obtener más información sobre el soporte de SQLj de DB2, visite la página Web situada en:

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

Conceptos relacionados:

- “Comparación entre SQLj y JDBC” en la página 284

Objetos de datos ActiveX y Objetos de datos remotos

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

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

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

Para ver ejemplos completos de aplicaciones DB2 que utilizan las especificaciones ADO y RDO, consulte los siguientes directorios:

- Para ejemplos de Objeto de datos ActiveX de Visual Basic, consulte `sqllib\samples\VB\ADO`
- Para ejemplos de Objeto de datos remotos de Visual Basic, consulte `sqllib\samples\VB\RDO`

- Para ejemplos de Servidor de transacciones Microsoft Visual Basic, consulte sqllib\samples\VB\MTS
- Para ejemplos de Objeto de datos ActiveX de Visual C++, consulte sqllib\samples\VC\ADO

Tareas relacionadas:

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

Consulta relacionada:

- "Programas de ejemplo Visual Basic" en el manual *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Programas de ejemplo Visual C++" en el manual *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

DBI Perl

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

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

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

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

Conceptos relacionados:

- "Consideraciones sobre la programación en Perl" en la página 363

Herramientas de usuario final de ODBC

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

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

Aplicaciones Web

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

Herramientas para crear aplicaciones Web

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

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

Conceptos relacionados:

- "WebSphere Studio" en la página 19
- "XML Extender" en la página 20

WebSphere Studio

WebSphere Studio es un grupo de herramientas que combinan todos los aspectos del desarrollo de sitios Web en una interfaz común. WebSphere Studio facilita más que nunca la creación, ensamblaje, publicación y mantenimiento en cooperación de aplicaciones Web interactivas. Studio consta de los componentes Workbench, Page Designer y Remote Debugger y asistentes y viene con copias de prueba de productos de desarrollo de Web

complementarios, como Macromedia Flash, Fireworks, Freehand y Director. WebSphere Studio le permite realizar todo lo que necesita para crear sitios Web interactivos que den soporte a funciones empresariales avanzadas.

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

WebSphere Application Server Advanced Edition (que no se suministra con DB2 Universal Developer's Edition) proporciona soporte adicional para aplicaciones Enterprise JavaBean. DB2 Universal Database se suministra con WebSphere Application Server Advanced Edition para que se utilice como el repositorio del servidor administrativo. Incorpora las funciones de servidor para aplicaciones creadas según la especificación EJB de Sun Microsystems, que proporciona soporte para la integración de aplicaciones Web en sistemas empresariales que no son Web.

Conceptos relacionados:

- "Enterprise Java Beans" en la página 348

Consulta relacionada:

- "Programas de ejemplo de Java WebSphere" en el manual *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

XML Extender

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

DB2 XML Extender incorpora tres nuevos tipos de datos: `XMLVARCHAR`, `XMLCLOB` y `XMLFILE`. El ampliador proporciona UDF para almacenar, extraer y actualizar documentos XML situados dentro de una sola columna o en varias columnas y tablas. La búsqueda se puede realizar en el documento XML entero o se

puede basar en componentes estructurales utilizando la vía de acceso de ubicación, que utiliza un subconjunto de Extensible Stylesheet Language Transformation (XSLT) y XPath para XML Path Language.

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

Para obtener más información sobre DB2 XML Extender, visite:

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

Conceptos relacionados:

- "Archivo de extensión de definición de acceso a documentos" en la página 343

Habilitación de MQSeries

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

En una configuración básica, un servidor MQSeries está situado en la máquina servidor de base de datos junto con DB2 Universal Database. Las funciones de MQSeries están disponibles desde un servidor DB2 y proporcionan acceso a otras aplicaciones MQSeries. Varios clientes DB2 pueden acceder simultáneamente a las funciones de MQSeries a través de la base de datos. Las operaciones de MQSeries permiten a las aplicaciones DB2 comunicarse de forma síncrona con otras aplicaciones MQSeries. Por ejemplo, las nuevas funciones proporcionan un método sencillo que permite a una aplicación DB2 publicar sucesos de bases de datos en aplicaciones MQSeries remotas, iniciar un flujo de trabajo a través del producto opcional MQSeries Workflow o comunicarse con un paquete de aplicación existente con el producto opcional MQSeries Integrator.

Net.Data

Net.Data permite el acceso de Internet e intranet a datos de DB2 a través de aplicaciones web. Aprovecha las interfaces (API) del servidor Web, proporcionando un rendimiento superior que las aplicaciones Common

Gateway Interface (CGI). Net.Data da soporte al proceso del cliente así como al proceso del servidor con lenguajes como Java, REXX, Perl y C++. Net.Data proporciona lógica adicional y un potente lenguaje de macros. También proporciona soporte de XML, lo que le permite generar códigos XML como salida de su macro Net.Data en lugar de tener que entrar los códigos de forma manual. También puede especificar una hoja de estilo XML (XSL) que se utilizará para formatear y visualizar la salida generada. Net.Data sólo está disponible si se baja de la Web. Para obtener más información, consulte el siguiente sitio Web:

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

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

Conceptos relacionados:

- "Herramientas para crear aplicaciones Web" en la página 19
- "XML Extender" en la página 20

Funciones de programación

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

Funciones de programación de DB2

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

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

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

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

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

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

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

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

Conceptos relacionados:

- "Procedimientos almacenados de DB2" en la página 23
- "Métodos y funciones definidas por el usuario de DB2" en la página 24
- "Tipos definidos por el usuario (UDT) y objetos grandes (LOB)" en la página 26
- "Activadores de DB2" en la página 29

Procedimientos almacenados de DB2

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

Puede obtener varias ventajas de la utilización de procedimientos almacenados:

Reducción del tráfico de la red

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

Acceso a funciones que sólo existen en el servidor

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

Cumplimiento de las normas empresariales

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

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

Conceptos relacionados:

- "Centro de desarrollo" en la página 25

Métodos y funciones definidas por el usuario de DB2

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

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

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

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

Centro de desarrollo

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

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

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

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

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

También puede iniciar el Centro de desarrollo desde el Centro de control para DB2 para OS/390. Puede iniciar el Centro de desarrollo como un proceso separado desde el menú Herramientas del Centro de control, desde la barra de herramientas o desde la carpeta Procedimientos almacenados. Asimismo, desde la ventana Proyecto del Centro de desarrollo puede exportar uno o más

procedimientos almacenados SQL creados para un servidor DB2 para OS/390 a un archivo específico que se pueda ejecutar en el procesador de línea de mandatos (CLP).

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

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

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

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

Conceptos relacionados:

- "Procedimientos almacenados de DB2" en la página 23
- "Rutinas de automatización de OLE" en la página 28

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

Cada elemento de datos de la base de datos se almacena en una columna de una tabla y cada columna se define con un tipo de datos. El tipo de datos impone límites en los tipos de valores que puede colocar en la columna y en las operaciones que puede realizar en los mismos. Por ejemplo, una columna de enteros sólo puede contener números comprendidos dentro de un rango

fijo. DB2 incluye un conjunto de tipos de datos integrados con características y comportamientos definidos: series de caracteres, numéricos, valores de fecha y hora, objetos grandes, valores nulos, series gráficas, series binarias y enlaces de datos.

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

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

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

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

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

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

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

(UDF), que se pueden escribir de modo que se acomoden a una amplia gama de requisitos del usuario.

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

Conceptos relacionados:

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

Rutinas de automatización de OLE

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

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

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

Conceptos relacionados:

- "Procedimientos almacenados de DB2" en la página 23
- "Centro de desarrollo" en la página 25

Funciones de tabla de OLE DB

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

La utilización de funciones de tabla OLE DB reduce el esfuerzo que hay que dedicar al desarrollo de aplicaciones al proporcionar acceso integrado a cualquier proveedor de OLE DB. Para funciones de tabla de automatización C, Java y OLE, el desarrollador tiene que implantar la función de tabla, mientras que en el caso de funciones de tabla OLE DB un consumidor genérico integrado OLE DB actúa como interfaz con cualquier proveedor de OLE DB para recuperar datos. Sólo tiene que registrar una función de tabla de tipo de lenguaje OLEDB y hacer referencia al proveedor de OLE DB y al conjunto de filas relevante como una fuente de datos. No tiene que realizar ninguna programación de UDF para aprovechar las funciones de tabla de OLE DB.

Conceptos relacionados:

- “Objetivo de IBM OLE DB Provider para DB2” en la página 391
- “Habilitación automática de servicios OLE DB por parte de IBM OLE DB Provider” en la página 396

Consulta relacionada:

- “Soporte de IBM OLE DB Provider de interfaces y componentes de OLE DB” en la página 403
- “Soporte de IBM OLE DB de propiedades de OLE DB” en la página 406

Activadores de DB2

Un activador define un conjunto de acciones ejecutadas por una operación de supresión, inserción o actualización en una tabla especificada. Cuando se ejecuta dicha operación de SQL, se dice que el activador se activa. El activador se puede activar antes de la operación de SQL o después de la misma. Puede definir un activador mediante la sentencia de SQL CREATE TRIGGER.

Puede utilizar activadores que se ejecuten antes de una actualización o inserción de varias formas:

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

Paralelamente, puede utilizar activadores que se ejecuten después de una actualización o inserción de varias formas:

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

La utilización de activadores presenta varias ventajas:

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

Cumplimiento global de las normas empresariales
Los activadores se definen una vez y los utilizan todas las aplicaciones que utilizan los datos controlados por los activadores.

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

Conceptos relacionados:

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

Capítulo 2. Codificación de una aplicación DB2

Requisitos previos para programación

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

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

Puede desarrollar aplicaciones en un servidor o en cualquier cliente que tenga instalado DB2 Application Development Client. Puede ejecutar aplicaciones con el servidor, DB2 Run-Time Client o DB2 Administrative Client. También puede desarrollar programas Java JDBC en uno de estos clientes, suponiendo que instale el componente ″Java Enablement″ cuando instale el cliente. Esto significa que puede ejecutar cualquier aplicación DB2 en estos clientes. Sin embargo, a no ser que también instale DB2 Application Development Client con estos clientes, sólo puede desarrollar aplicaciones JDBC en los mismos.

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

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

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

Tareas relacionadas:

- “Configuración del entorno de desarrollo de aplicaciones” en el manual *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- “Configuración de la base de datos sample” en el manual *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

Visión general de la codificación de aplicaciones DB2

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

Programación de una aplicación autónoma

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

Procedimiento:

Para programar una aplicación autónoma, debe asegurarse de:

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

Conceptos relacionados:

- “Requisitos previos para programación” en la página 32
- “Infraestructura de seudocódigo de aplicación” en la página 49
- “Recursos para crear prototipos de sentencias de SQL” en la página 50
- “Archivos de ejemplo” en el manual *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- “Programas de ejemplo: estructura y diseño” en el manual *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

Tareas relacionadas:

- “Creación de una sección de declaración de una aplicación autónoma” en la página 34
- “Conexión de una aplicación con una base de datos” en la página 43
- “Codificación de transacciones” en la página 44
- “Finalización de una transacción con la sentencia COMMIT” en la página 45
- “Finalización de una transacción con la sentencia ROLLBACK” en la página 46

- “Finalización de un programa de aplicación” en la página 48
- “Configuración de un entorno de prueba” en la página 64

Creación de una sección de declaración de una aplicación autónoma

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

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

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

Un programa puede contener varias secciones declare de SQL.

Procedimiento:

Para crear la sección de declaración:

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

Conceptos relacionados:

- “Valores de SQLSTATE y SQLCODE en Java” en la página 335

Tareas relacionadas:

- “Declaración de variables que interactúan con el gestor de bases de datos” en la página 34
- “Declaración de variables que representan objetos de SQL” en la página 35
- “Relación de variables del lenguaje principal con una sentencia de SQL” en la página 39
- “Declaración de variables del lenguaje principal con el Generador de declaraciones db2dclgn” en la página 38
- “Declaración de la SQLCA para el manejo de errores” en la página 40

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

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

Las variables de programa de sistema principal declaradas en una sección de declaración de SQL se denominan variables del lenguaje principal. Puede utilizar variables del lenguaje principal en referencias a variables del lenguaje principal en sentencias de SQL. El código *variable-lenguaje-principal* se utiliza en diagramas de sintaxis en sentencias de SQL.

Procedimiento:

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

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

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

Consulta relacionada:

- “Tipos de datos SQL soportados en C y C++” en la página 218
- “Tipos de datos de SQL soportados en COBOL” en la página 254
- “Tipos de datos SQL soportados en FORTRAN” en la página 276
- “Tipos de datos SQL soportados en Java” en la página 291
- “Tipos de datos SQL soportados en REXX” en la página 381

Declaración de variables que representan objetos de SQL

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

Procedimiento:

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

Cuando codifique la variable, recuerde que los nombres de tablas, alias, vistas y correlaciones tienen una longitud máxima de 128 bytes. Los nombres de

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

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

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

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

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

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

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

Conceptos relacionados:

- “Variables del lenguaje principal en C y C++” en la página 183
- “Sintaxis de las variables del lenguaje principal de tipo carácter fijas y terminadas en nulo en C y C++” en la página 189

- “Expansión de macros en C” en la página 200
- “Variables del lenguaje principal en COBOL” en la página 240
- “Variables del lenguaje principal en FORTRAN” en la página 269
- “Variables del lenguaje principal en Java” en la página 290
- “Variables del lenguaje principal en REXX” en la página 374

Consulta relacionada:

- “Sintaxis de las variables numéricas del lenguaje principal en C y C++” en la página 187
- “Sintaxis de las variables del lenguaje principal de tipo carácter de longitud variable en C o C++” en la página 190
- “Sintaxis de la declaración gráfica de formatos gráficos de un solo gráfico y terminados en nulo en C y C++” en la página 192
- “Sintaxis para la declaración gráfica del formato estructurado VARGRAPHIC en C o C++” en la página 194
- “Sintaxis de las variables del lenguaje principal de objeto grande (LOB) en C o C++” en la página 195
- “Sintaxis de las variables del lenguaje principal de localizador de objeto grande (LOB) en C o C++” en la página 198
- “Sintaxis de declaraciones de variables del lenguaje principal de referencia de archivos en C o C++” en la página 199
- “Sintaxis de las variables numéricas del lenguaje principal en COBOL” en la página 242
- “Sintaxis de las variables del lenguaje principal de tipo carácter de longitud fija en COBOL” en la página 243
- “Sintaxis de las variables gráficas del lenguaje principal de longitud fija en COBOL” en la página 245
- “Sintaxis de las variables del lenguaje principal LOB en COBOL” en la página 246
- “Sintaxis de las variables del lenguaje principal de localizador de LOB en COBOL” en la página 248
- “Sintaxis de las variables del lenguaje principal de referencia de archivos en COBOL” en la página 248
- “Sintaxis de las variables numéricas del lenguaje principal en FORTRAN” en la página 271
- “Sintaxis de las variables de tipo carácter del lenguaje principal en FORTRAN” en la página 271
- “Sintaxis de las variables del lenguaje principal de objeto grande (LOB) en FORTRAN” en la página 273
- “Sintaxis de las variables del lenguaje principal de localizador de objeto grande (LOB) en FORTRAN” en la página 274

- “Sintaxis de las variables del lenguaje principal de referencia de archivos en FORTRAN” en la página 275
- “Sintaxis de las declaraciones de localizador de LOB en REXX” en la página 378
- “Sintaxis de las declaraciones de referencia de archivos LOB en REXX” en la página 379

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

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

Procedimiento:

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

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

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

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

El archivo staff.h resultante contiene:

```
struct
{
  short id;
  struct
  {
    short length;
    char data[9];
  } name;
  short dept;
  char job[5];
  short years;
  double salary;
  double comm;
} staff;
```

Consulta relacionada:

- “db2dclgn - Mandato Generador de declaraciones” en el manual *Consulta de mandatos*

Relación de variables del lenguaje principal con una sentencia de SQL

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

Considere la siguiente sentencia SELECT INTO:

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

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

Procedimiento:

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

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

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

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

   Java:
   ```
   int variable_name;
   ```

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

FORTRAN:

```
INTEGER*4 variable_name
```

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

Conceptos relacionados:

- "Vistas de catálogo" en el manual *Consulta de SQL, Volumen 1*

Tareas relacionadas:

- "Declaración de variables que interactúan con el gestor de bases de datos" en la página 34
- "Declaración de variables del lenguaje principal con el Generador de declaraciones db2dclgn" en la página 38
- "Creación de una sección de declaración de una aplicación autónoma" en la página 34

Consulta relacionada:

- "Tipos de datos SQL soportados en C y C++" en la página 218
- "Tipos de datos de SQL soportados en COBOL" en la página 254
- "Tipos de datos SQL soportados en FORTRAN" en la página 276
- "Tipos de datos SQL soportados en Java" en la página 291
- "Tipos de datos SQL soportados en REXX" en la página 381

Declaración de la SQLCA para el manejo de errores

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

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

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

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

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

Procedimiento:

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

- Para aplicaciones C o C++ utilice:

  ```
  EXEC SQL INCLUDE SQLCA;
  ```

- En aplicaciones Java no se utiliza de forma explícita la SQLCA. Utilice en su lugar los métodos de instancia `SQLException` para obtener los valores de SQLSTATE y SQLCODE.
- Para aplicaciones COBOL utilice:

  ```
  EXEC SQL INCLUDE SQLCA END-EXEC.
  ```

- Para aplicaciones FORTRAN utilice:

  ```
  EXEC SQL INCLUDE SQLCA
  ```

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

Conceptos relacionados:

- “Definición de FIPS 127-2 e ISO/ANS SQL92” en la página 52
- “Manejo de errores utilizando la sentencia WHENEVER” en la página 41
- “Variables SQLSTATE y SQLCODE en C y C++” en la página 224
- “Variables SQLSTATE y SQLCODE en COBOL” en la página 257
- “Variables SQLSTATE y SQLCODE en FORTRAN” en la página 279
- “Valores de SQLSTATE y SQLCODE en Java” en la página 335
- “Variables SQLSTATE y SQLCODE en Perl” en la página 366

Tareas relacionadas:

- “Creación de una sección de declaración de una aplicación autónoma” en la página 34

Manejo de errores utilizando la sentencia WHENEVER

La sentencia WHENEVER hace que el precompilador genere código fuente que indica a la aplicación que vaya a una etiqueta especificada si se produce un error, un aviso o si no se encuentran filas durante la ejecución. La

sentencia WHENEVER afecta a las siguientes sentencias de SQL ejecutables hasta que otra sentencia WHENEVER altera la situación.

La sentencia WHENEVER tiene tres formas básicas:

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

En las sentencias anteriores:

SQLERROR
Identifica cualquier condición en la que SQLCODE < 0.

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

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

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

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

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

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

La sentencia WHENEVER debe aparecer antes de las sentencias de SQL que desea que se vean afectadas. De lo contrario, el precompilador no sabe que se debe generar código de manejo de errores adicional para las sentencias de SQL ejecutables. Puede tener cualquier combinación de las tres formas básicas activas en cualquier momento. El orden en el que declare las tres formas no es significativo.

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

Consulta relacionada:

- "WHENEVER sentencia" en el manual *Consulta de SQL, Volumen 2*

Adición de sentencias no ejecutables a una aplicación

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

Procedimiento:

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

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

Tareas relacionadas:

- "Creación de una sección de declaración de una aplicación autónoma" en la página 34

Conexión de una aplicación con una base de datos

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

Restricciones:

Se aplican las siguientes restricciones:

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

Procedimiento:

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

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

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

Conceptos relacionados:

- "Actualización múltiple" en la página 461

Consulta relacionada:

- "CONNECT (Tipo 1) sentencia" en el manual *Consulta de SQL, Volumen 2*
- "CONNECT (Tipo 2) sentencia" en el manual *Consulta de SQL, Volumen 2*

Codificación de transacciones

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

Requisitos previos:

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

Procedimiento:

Para codificar una transacción:

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

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

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

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

Nota: las seis sentencias siguientes *no* inician una transacción porque no son sentencias ejecutables:

- BEGIN DECLARE SECTION
- INCLUDE SQLCA
- END DECLARE SECTION
- INCLUDE SQLDA
- DECLARE CURSOR
- WHENEVER

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

Tareas relacionadas:

- "Finalización de una transacción con la sentencia COMMIT" en la página 45
- "Finalización de una transacción con la sentencia ROLLBACK" en la página 46

Finalización de una transacción con la sentencia COMMIT

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

Procedimiento:

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

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

Si no finaliza las transacciones de forma explícita, DB2 confirma automáticamente todos los cambios realizados durante la transacción pendiente del programa cuando el programa finaliza satisfactoriamente,

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

DB2 retrotrae los cambios bajo las siguientes condiciones:

- Una condición de registro lleno
- Cualquier otra condición del sistema que haga que finalice el proceso del gestor de bases de datos

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

Conceptos relacionados:

- "Finalización implícita de una transacción en una aplicación autónoma" en la página 48
- "Códigos de retorno" en la página 134
- "Información de error en los campos SQLCODE, SQLSTATE y SQLWARN" en la página 134

Tareas relacionadas:

- "Finalización de un programa de aplicación" en la página 48

Consulta relacionada:

- "COMMIT sentencia" en el manual *Consulta de SQL, Volumen 2*

Finalización de una transacción con la sentencia ROLLBACK

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

Procedimiento:

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

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

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

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

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

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

Conceptos relacionados:

- “Finalización implícita de una transacción en una aplicación autónoma” en la página 48
- “Unidad de trabajo remota” en la página 461
- “Actualización múltiple” en la página 461

Consulta relacionada:

- “CONNECT (Tipo 1) sentencia” en el manual *Consulta de SQL, Volumen 2*
- “CONNECT (Tipo 2) sentencia” en el manual *Consulta de SQL, Volumen 2*
- “WHENEVER sentencia” en el manual *Consulta de SQL, Volumen 2*

Finalización de un programa de aplicación

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

Procedimiento:

Para finalizar correctamente el programa:

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

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

Conceptos relacionados:

- "Finalización implícita de una transacción en una aplicación autónoma" en la página 48

Consulta relacionada:

- "CONNECT (Tipo 1) sentencia" en el manual *Consulta de SQL, Volumen 2*
- "CONNECT (Tipo 2) sentencia" en el manual *Consulta de SQL, Volumen 2*

Finalización implícita de una transacción en una aplicación autónoma

Si el programa termina sin finalizar la transacción actual, DB2 termina de forma implícita la transacción actual. DB2 termina de forma implícita la transacción actual emitiendo una sentencia COMMIT o ROLLBACK y luego la aplicación finaliza. Si la emisión de una sentencia COMMIT o ROLLBACK por parte de DB2 depende de factores como los siguientes:

- Si la aplicación ha terminado normalmente

  En la mayoría de los sistemas operativos soportados, DB2 confirma de forma implícita una transacción si la terminación es normal o retrotrae de forma implícita la transacción si la terminación es anómala. Tenga en cuenta que lo que el programa considera una terminación anómala puede no ser considerado como tal por el gestor de bases de datos. Por ejemplo, puede codificar `exit(-16)` cuando la aplicación encuentre un error inesperado y

terminar la aplicación de inmediato. El gestor de bases de datos considera que esto es una terminación normal y confirma la transacción. El gestor de bases de datos considera elementos tales como una excepción o una infracción de segmentación como terminaciones anómalas.

- La plataforma en la que se ejecuta el servidor DB2

  En sistemas operativos Windows de 32 bits, DB2 siempre retrotrae la transacción, independientemente de si la aplicación termina de forma normal o anómala. Si desea que la transacción se confirme, debe emitir la sentencia COMMIT de forma explícita.

- Si la aplicación utiliza las API de contexto de DB2 para el acceso a bases de datos de varias hebras

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

Conceptos relacionados:

- "Objetivo del acceso a base de datos de varias hebras" en la página 227

Tareas relacionadas:

- "Finalización de un programa de aplicación" en la página 48

Consulta relacionada:

- "COMMIT sentencia" en el manual *Consulta de SQL, Volumen 2*
- "ROLLBACK sentencia" en el manual *Consulta de SQL, Volumen 2*

Infraestructura de seudocódigo de aplicación

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

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

  (lógica de programa)

EXEC SQL CONNECT TO base_de_datos A USER :id_usuario USING :pw          |
EXEC SQL SELECT ...                                             |
EXEC SQL INSERT ...                                             | Primera unidad
```

```
  (más sentencias de SQL)                                    | de trabajo
EXEC SQL COMMIT                                              |

  (más lógica de programa)

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

  (más lógica de programa)

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

  (más lógica de programa)

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

Finalizar programa
```

Tareas relacionadas:

- “Programación de una aplicación autónoma” en la página 33

Recursos para crear prototipos de sentencias de SQL

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

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

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

También puede redirigir la entrada y la salida de las peticiones del procesador de línea de mandatos. Por ejemplo, podría crear uno o más archivos que contengan sentencias de SQL que necesita como entrada en una petición del procesador de línea de mandatos para no tener que volver a escribir la sentencia.

- Utilizar el recurso Explain para obtener una idea de los costes estimados de las sentencias DELETE, INSERT, UPDATE o SELECT que tiene intención de utilizar en el programa. El recurso Explain coloca la información sobre la estructura y los costes estimados de la sentencia en cuestión en tablas suministradas por el usuario. Puede ver esta información utilizando Visual Explain o el programa de utilidad `db2exfmt`.
- Utilizar las vistas del catálogo del sistema para recuperar fácilmente información sobre las bases de datos existentes. El gestor de bases de datos crea y mantiene las tablas del catálogo del sistema en las que se basan las vistas durante el funcionamiento normal a medida que las bases de datos se crean, modifican y actualizan. Estas vistas contienen datos sobre cada base de datos, que incluyen autorizaciones otorgadas, nombres de columna, tipos de datos, índices, dependencias de paquetes, restricciones referenciales, nombres de tabla, vistas, etc. Los datos de las vistas del catálogo del sistema están disponibles a través de los recursos de consulta normales de SQL.

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

Conceptos relacionados:

- “Vistas de catálogo” en el manual *Consulta de SQL, Volumen 1*
- “Catalog statistics tables” en el manual *Administration Guide: Performance*
- “Catalog statistics for modeling and what-if planning” en el manual *Administration Guide: Performance*
- “General rules for updating catalog statistics manually” en el manual *Administration Guide: Performance*
- “SQL explain facility” en el manual *Administration Guide: Performance*
- “Herramientas de DB2 Universal Database para desarrollar aplicaciones” en la página 5

Consulta relacionada:

- Apéndice A, “Sentencias de SQL soportadas” en la página 521

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

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

Puede utilizar las API de DB2 para:

- Manipular el entorno del gestor de bases de datos, que incluye la catalogación y descatalogación de bases de datos y nodos y la exploración de directorios de bases de datos y de nodos. También puede utilizar API para crear, suprimir y migrar bases de datos.
- Proporcionar recursos para importar y exportar datos y administrar, hacer copia de seguridad y restaurar la base de datos.
- Modificar los valores de parámetros de configuración del gestor de bases de datos y de la base de datos.
- Proporcionar operaciones específicas del entorno cliente/servidor.
- Proporcionar la interfaz en tiempo de ejecución para sentencias de SQL precompiladas. El desarrollador no suele llamar directamente a estas API. En su lugar, el precompilador las inserta en el archivo fuente modificado después del proceso.

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

Conceptos relacionados:

- "Consideraciones sobre autorización para API" en la página 63
- "Programas de ejemplo: estructura y diseño" en el manual *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

Definición de FIPS 127-2 e ISO/ANS SQL92

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

Control de valores de datos y relaciones

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

Control de valores de datos

Un área tradicional de la lógica de aplicación es validar y proteger la integridad de los datos, controlando los valores permitidos en la base de

datos. Las aplicaciones tienen lógica que comprueba de forma específica la validez de los valores de datos a medida que se entran. Por ejemplo, comprobación de que el número de departamento es un número válido y de que hace referencia a un departamento existente. Hay varias formas de proporcionar estas mismas funciones en DB2, pero desde dentro de la base de datos.

Conceptos relacionados:

- "Control de valores de datos mediante tipos de datos" en la página 53
- "Control de valores de datos mediante restricciones exclusivas" en la página 53
- "Control de valores de datos mediante restricciones de comprobación de tabla" en la página 54
- "Control de valores de datos mediante restricciones de integridad referencial" en la página 54
- "Control de valores de datos mediante vistas con la opción Check" en la página 55
- "Control de valores de datos mediante lógica de aplicación y tipos de variables de programa" en la página 55

Control de valores de datos mediante tipos de datos

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

Conceptos relacionados:

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

Control de valores de datos mediante restricciones exclusivas

Las restricciones exclusivas evitan la presencia de valores duplicados en una o más columnas dentro de una tabla. Las claves exclusivas y primarias son las

restricciones exclusivas soportadas. Por ejemplo, puede definir una restricción exclusiva en la columna DEPTNO de la tabla DEPARTMENT para asegurar que no se asigna el mismo número de departamento a dos departamentos.

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

Tareas relacionadas:

- "Defining a unique constraint" en el manual *Administration Guide: Implementation*
- "Adding a unique constraint" en el manual *Administration Guide: Implementation*

Control de valores de datos mediante restricciones de comprobación de tabla

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

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

Tareas relacionadas:

- "Defining a table check constraint" en el manual *Administration Guide: Implementation*
- "Adding a table check constraint" en el manual *Administration Guide: Implementation*

Control de valores de datos mediante restricciones de integridad referencial

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

Conceptos relacionados:

- "Restricciones" en el manual *Consulta de SQL, Volumen 1*
- "Control de relaciones de datos mediante restricciones de integridad referencial" en la página 56
- "Diferencias en la integridad referencial entre sistemas de bases de datos relacionales de IBM" en la página 535

Control de valores de datos mediante vistas con la opción Check

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

Consulta relacionada:

- "CREATE VIEW sentencia" en el manual *Consulta de SQL, Volumen 2*

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

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

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

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

Conceptos relacionados:

- "Control de valores de datos mediante vistas con la opción Check" en la página 55

Control de relaciones de datos

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

Conceptos relacionados:

- "Control de relaciones de datos mediante restricciones de integridad referencial" en la página 56
- "Control de relaciones de datos mediante activadores" en la página 57
- "Control de relaciones de datos mediante activadores anteriores" en la página 57
- "Control de relaciones de datos mediante activadores posteriores" en la página 58
- "Control de relaciones de datos mediante lógica de aplicación" en la página 58

Control de relaciones de datos mediante restricciones de integridad referencial

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

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

Conceptos relacionados:

- "Restricciones" en el manual *Consulta de SQL, Volumen 1*

Tareas relacionadas:

- "Defining referential constraints" en el manual *Administration Guide: Implementation*

Consulta relacionada:

- "ALTER TABLE sentencia" en el manual *Consulta de SQL, Volumen 2*

- "CREATE TABLE sentencia" en el manual *Consulta de SQL, Volumen 2*

Control de relaciones de datos mediante activadores

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

Conceptos relacionados:

- "Control de relaciones de datos mediante activadores anteriores" en la página 57
- "Control de relaciones de datos mediante activadores posteriores" en la página 58
- "Activadores de DB2" en la página 29

Tareas relacionadas:

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

Consulta relacionada:

- "CREATE TRIGGER sentencia" en el manual *Consulta de SQL, Volumen 2*

Control de relaciones de datos mediante activadores anteriores

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

Conceptos relacionados:

- "Control de relaciones de datos mediante activadores posteriores" en la página 58
- "Activadores de DB2" en la página 29

Tareas relacionadas:

- "Creating a trigger" en el manual *Administration Guide: Implementation*

- "Creación de activadores" en el manual *Guía de desarrollo de aplicaciones: Programación de aplicaciones de servidor*

Consulta relacionada:

- "CREATE TRIGGER sentencia" en el manual *Consulta de SQL, Volumen 2*

Control de relaciones de datos mediante activadores posteriores

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

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

Conceptos relacionados:

- "Control de relaciones de datos mediante activadores anteriores" en la página 57
- "Activadores de DB2" en la página 29

Tareas relacionadas:

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

Consulta relacionada:

- "CREATE TRIGGER sentencia" en el manual *Consulta de SQL, Volumen 2*

Control de relaciones de datos mediante lógica de aplicación

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

Conceptos relacionados:

- “Lógica de aplicación en el servidor” en la página 59

Lógica de aplicación en el servidor

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

Puede utilizar lo siguiente:

- Procedimientos almacenados

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

- Funciones definidas por el usuario

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

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

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

- activadores

  Se pueden utilizar activadores para invocar funciones definidas por el usuario. Esto resulta útil si desea que siempre se lleve a cabo una determinada operación no SQL cuando se produzcan determinadas sentencias o se cambien valores de datos. Ejemplos tales son operaciones como emitir un mensaje de correo electrónico bajo circunstancias específicas o grabar información de tipo de alerta en un archivo.

Conceptos relacionados:

- "Control de relaciones de datos mediante activadores anteriores" en la página 57
- "Control de relaciones de datos mediante activadores posteriores" en la página 58
- "Guidelines for stored procedures" en el manual *Administration Guide: Performance*
- "Interacciones de los activadores con las restricciones referenciales" en el manual *Guía de desarrollo de aplicaciones: Programación de aplicaciones de servidor*
- "Procedimientos almacenados de DB2" en la página 23
- "Métodos y funciones definidas por el usuario de DB2" en la página 24
- "Activadores de DB2" en la página 29

Tareas relacionadas:

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

Consulta relacionada:

- "CREATE TRIGGER sentencia" en el manual *Consulta de SQL, Volumen 2*

Consideraciones sobre autorización para SQL y API

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

Consideraciones sobre autorización para SQL incorporado

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

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

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

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

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

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

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

Conceptos relacionados:

- “Consideraciones sobre autorización para SQL dinámico” en la página 61
- “Consideraciones sobre autorización para SQL estático” en la página 63
- “Consideraciones sobre autorización para API” en la página 63
- “Authorization” en el manual *Administration Guide: Planning*

Consideraciones sobre autorización para SQL dinámico

Para utilizar SQL dinámico en un paquete vinculado con DYNAMICRULES RUN (valor por omisión), la persona que ejecuta una aplicación de SQL dinámico debe tener los privilegios necesarios para emitir cada petición de SQL realizada, así como el privilegio EXECUTE sobre el paquete. Los

privilegios se pueden otorgar al ID de autorización del usuario, a cualquier grupo del que el usuario sea miembro o a PUBLIC.

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

Si el programa no contiene SQL estático, la persona que vincula la aplicación (para aplicaciones de SQL dinámico incorporado) sólo necesita la autorización BINDADD sobre la base de datos. De nuevo, este privilegio se puede otorgar al ID de autorización del usuario, a un grupo del que el usuario sea miembro o a PUBLIC.

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

Conceptos relacionados:

- “Consideraciones sobre autorización para SQL incorporado” en la página 60
- “Consideraciones sobre autorización para SQL estático” en la página 63
- “Consideraciones sobre autorización para API” en la página 63
- “Efectos de DYNAMICRULES en SQL dinámico” en la página 147

Consulta relacionada:

- “Mandato BIND” en el manual *Consulta de mandatos*

Consideraciones sobre autorización para SQL estático

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

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

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

Conceptos relacionados:

- “Consideraciones sobre autorización para SQL incorporado” en la página 60
- “Consideraciones sobre autorización para SQL dinámico” en la página 61
- “Consideraciones sobre autorización para API” en la página 63

Consulta relacionada:

- “Mandato BIND” en el manual *Consulta de mandatos*

Consideraciones sobre autorización para API

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

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

Conceptos relacionados:

- "Consideraciones sobre autorización para SQL incorporado" en la página 60
- "Consideraciones sobre autorización para SQL dinámico" en la página 61
- "Consideraciones sobre autorización para SQL estático" en la página 63

Prueba de la aplicación

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

Configuración del entorno de prueba para una aplicación

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

Configuración de un entorno de prueba

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

Procedimiento:

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

1. Cree una base de datos de prueba.

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

2. Cree tablas y vistas de prueba.

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

3. Genere datos de prueba para las tablas.

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

4. Depure y optimice la aplicación.

Tareas relacionadas:

- "Creación de tablas y vistas de prueba" en la página 65
- "Generación de datos de prueba" en la página 66
- "Depuración y optimización de una aplicación" en la página 68

Consulta relacionada:

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

Creación de tablas y vistas de prueba

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

Procedimiento:

Para crear tablas de prueba:

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

Tabla 1. Descripción de los datos de la aplicación

Nombre de tabla o vista	Insertar filas	Suprimir filas	Nombre columna	Tipo de datos	Acceso de actualización
TEST.TEMPL	No	No	EMPNO	CHAR(6)	Sí
			LASTNAME	VARCHAR(15)	Sí
			WORKDEPT	CHAR(3)	Sí
			PHONENO	CHAR(4)	
			JOBCODE	DECIMAL(3)	
TEST.TDEPT	No	No	DEPTNO	CHAR(3)	
			MGRNO	CHAR(6)	
TEST.TPROJ	Sí	Sí	PROJNO	CHAR(6)	Sí
			DEPTNO	CHAR(3)	Sí
			RESPEMP	CHAR(6)	Sí
			PRSTAFF	DECIMAL(5,2)	Sí
			PRSTDATE	DECIMAL(6)	Sí
			PRENDATE	DECIMAL(6)	

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

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

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

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

Tareas relacionadas:

- “Generación de datos de prueba” en la página 66

Consulta relacionada:

- “CREATE TABLE sentencia” en el manual *Consulta de SQL, Volumen 2*

Generación de datos de prueba

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

Procedimiento:

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

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

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

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

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

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

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

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

SELECT * FROM EMP;
```

A continuación se explica la sentencia anterior:

1. La primera parte de la sentencia INSERT genera 100 registros correspondientes a los 100 primeros empleados utilizando una subconsulta

recursiva para generar los números de empleado. Cada registro contiene el número de empleado. Para cambiar el número de empleados, utilice un número distinto de 100.

2. La sentencia SELECT genera la columna LASTNAME. Empieza generando un entero aleatorio de hasta 6 dígitos de longitud utilizando la función RAND. Luego convierte el entero en su formato de carácter numérico utilizando la función CHAR.
3. Para convertir los caracteres numéricos en caracteres alfabéticos, la sentencia utiliza la función TRANSLATE para convertir los diez caracteres numéricos (de 0 a 9) en caracteres alfabéticos. Puesto que hay más de 10 caracteres alfabéticos, la sentencia selecciona entre cinco conversiones diferentes. Esto da como resultado nombres que tienen suficientes vocales aleatorias para que se puedan pronunciar y por eso las vocales se incluyen en cada conversión.
4. La sentencia genera un valor de HIREDATE aleatorio. El valor de HIREDATE va, hacia atrás, desde la fecha actual hasta hace 30 años. HIREDATE se calcula restando un número aleatorio de días, comprendido entre 0 y 10.957, de la fecha actual. (10.957 es el número de días que hay en 30 años.)
5. Finalmente, la sentencia genera aleatoriamente el valor de SALARY. El salario mínimo es 10.000, al que se suma un número aleatorio comprendido entre 0 y 200.000.

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

Conceptos relacionados:

- "Import Overview" en el manual *Data Movement Utilities Guide and Reference*
- "Load Overview" en el manual *Data Movement Utilities Guide and Reference*
- "Métodos y funciones definidas por el usuario de DB2" en la página 24

Tareas relacionadas:

- "Depuración y optimización de una aplicación" en la página 68

Consulta relacionada:

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

Depuración y optimización de una aplicación

Puede depurar y optimizar la aplicación mientras la desarrolla.

Procedimiento:

Para depurar y optimizar la aplicación:

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

Conceptos relacionados:

- "Catalog statistics for modeling and what-if planning" en el manual *Administration Guide: Performance*
- "Recursos para crear prototipos de sentencias de SQL" en la página 50
- "The database system-monitor information" en el manual *Administration Guide: Performance*
- "Requisitos del archivo fuente para aplicaciones de SQL incorporado" en la página 86

Macro automática de proyectos de IBM DB2 Universal Database para Microsoft Visual C++

Las secciones siguientes describen la macro automática de proyectos de IBM DB2 Universal Database para Microsoft Visual C++.

La macro automática de proyectos de IBM DB2 Universal Database para Microsoft Visual C++

La macro automática de proyectos de IBM DB2 Universal Database para Microsoft Visual C++ es una serie de asistentes y herramientas de gestión que se conectan al componente Visual C++ del IDE Visual Studio. Las herramientas y asistentes automatizan y simplifican varias tareas que se deben llevar a cabo para desarrollar aplicaciones para DB2 que utiliza SQL incorporado.

Puede utilizar la macro automática de proyectos de IBM DB2 Universal Database para Microsoft Visual C++ para desarrollar, empaquetar y desplegar:

- Procedimientos almacenados escritos en C/C++ para DB2 Universal Database en sistemas operativos Windows
- Aplicaciones cliente de SQL incorporado de Windows C/C++ que acceden a servidores DB2 Universal Database
- Aplicaciones cliente de Windows C/C++ que invocan procedimientos almacenados utilizando reiniciadores de llamada de función C/C++

La macro automática de proyectos de IBM DB2 Universal Database para Microsoft Visual C++ le permite centrarse en el diseño y en la lógica de su aplicación DB2 en lugar de tenerse que preocupar de la creación y despliegue de la misma.

Algunas de las tareas que lleva a cabo la macro automática de proyectos de IBM DB2 Universal Database para Microsoft Visual C++ incluyen:

- Creación de un nuevo módulo de SQL incorporado
- Inserción de sentencias de SQL en un módulo de SQL incorporado utilizando SQL Assist
- Adición de procedimientos almacenados importados
- Creación de un procedimiento almacenado exportado
- Empaquetado del proyecto de DB2
- Despliegue del proyecto de DB2 desde dentro de Visual C++

La macro automática de proyectos de IBM DB2 Universal Database para Microsoft Visual C++ se presenta en forma de una barra de herramientas. Los botones de la barra de herramientas incluyen:

Propiedades de proyecto de DB2
Gestiona las propiedades del proyecto (opciones de base de datos de desarrollo y de generación de código)

Nuevo objeto de DB2
Añade un nuevo módulo de SQL incorporado, procedimiento almacenado importado o procedimiento almacenado exportado

Módulos de SQL incorporado de DB2
Gestiona la lista de módulos de SQL incorporado y sus opciones de precompilador

Procedimientos almacenados importados de DB2
Gestiona la lista de procedimientos almacenados importados

Procedimientos almacenados exportados de DB2
Gestiona la lista de procedimientos almacenados exportados

Empaquetar proyecto de DB2
Empaqueta los archivos externos de proyecto de DB2

Desplegar proyecto de DB2
Despliega los archivos externos de proyecto de DB2 empaquetado

La macro automática de proyectos de IBM DB2 Universal Database para Microsoft Visual C++ también tiene los tres siguientes botones ocultos, que pueden dejarse visibles utilizando las opciones estándares de personalización de las herramientas de Visual C++:

Nuevo módulo de SQL incorporado de DB2
Añade un nuevo módulo de SQL incorporado de C/C++

Nuevo procedimiento almacenado importado de DB2
Importa un nuevo procedimiento almacenado de base de datos

Nuevo procedimiento almacenado exportado de DB2
Exporta un nuevo procedimiento almacenado de base de datos

La macro automática de proyectos de IBM DB2 Universal Database para Microsoft Visual C++ puede generar automáticamente los siguientes elementos de código:

- Archivos de módulo de SQL incorporado que forman el esqueleto con sentencias de SQL opcionales de ejemplo
- Funciones estándares de SQL incorporado de conexión y desconexión de base de datos
- Funciones de reiniciador de llamada de procedimiento almacenado importado
- Plantillas de función de procedimiento almacenado exportado
- Archivos de lenguaje de definición de datos (DDL) de procedimiento almacenado exportado

Para obtener más información sobre la Macro automática de proyectos de IBM DB2 Universal Database para Microsoft Visual C++, consulte:

- La ayuda en línea correspondiente a la macro automática de proyectos de IBM DB2 Universal Database para Microsoft Visual C++
- http://www.ibm.com/software/data/db2/udb/ide/index.html

Tareas relacionadas:

- “Activación de la macro automática de proyectos de IBM DB2 Universal Database para Microsoft Visual C++” en la página 73
- “Activación de la macro automática de herramientas de IBM DB2 Universal Database para Microsoft Visual C++” en la página 74

Consulta relacionada:

- “Terminología de la macro automática de proyectos de IBM DB2 Universal Database para Microsoft Visual C++” en la página 72

Terminología de la macro automática de proyectos de IBM DB2 Universal Database para Microsoft Visual C++

La terminología asociada con la macro automática de proyectos de IBM DB2 Universal Database para Microsoft Visual C++ es la siguiente:

proyecto de IDE
El proyecto estándar Visual C++

proyecto de DB2
EL grupo de objetos de proyecto de DB2 que se insertan en el proyecto de IDE. Los objetos de proyecto de DB2 se pueden insertar en cualquier proyecto de Visual C++. El proyecto de DB2 le permite gestionar diversos objetos de DB2, como módulos de SQL incorporado, procedimientos almacenados importados y procedimientos almacenados exportados. Puede añadir, suprimir y modificar estos objetos y sus propiedades.

módulo
Un archivo en código fuente C/C++ que puede contener sentencias de SQL.

base de datos de desarrollo
La base de datos que sirve para compilar módulos de SQL intercalado. La base de datos de desarrollo también sirve para buscar la lista de definiciones de procedimientos almacenados de base de datos que se pueden importar.

módulo de SQL incorporado
Un archivo en código fuente C/C++ que contiene SQL dinámico o estático incorporado.

procedimiento almacenado importado
Un procedimiento almacenado, ya definido en la base de datos, que invoca el proyecto.

procedimiento almacenado exportado
Un procedimiento almacenado de base de datos que crea y define el proyecto.

Conceptos relacionados:

- “La macro automática de proyectos de IBM DB2 Universal Database para Microsoft Visual C++” en la página 69

Tareas relacionadas:

- “Activación de la macro automática de proyectos de IBM DB2 Universal Database para Microsoft Visual C++” en la página 73
- “Activación de la macro automática de herramientas de IBM DB2 Universal Database para Microsoft Visual C++” en la página 74

Activación de la macro automática de proyectos de IBM DB2 Universal Database para Microsoft Visual C++

Active la macro automática de proyectos de IBM DB2 Universal Database para Microsoft Visual C++ para acceder a la barra de herramientas flotante.

Nota: si la barra de herramientas se cierra accidentalmente, puede desactivar y volver a activar la macro automática o utilizar las opciones de personalización estándar de Microsoft Visual C++ para volver a visualizar la barra de herramientas.

Procedimiento:

1. Inicie y detenga Visual C++ al menos una vez con su ID de conexión actual. La primera vez que se ejecuta Visual C++, se crea un perfil para el ID de usuario, y esto es lo que se actualiza mediante el mandato `db2vccmd`. Si no lo ha iniciado una vez e intenta ejecutar `db2vccmd`, se puede encontrar errores como los siguientes:

   ```
   "Registering DB2 Project add-in ...Failed! (rc = 2)"
   ```

2. Registre la macro automática, si aún no la ha hecho, especificando lo siguiente en la línea de mandatos:

   ```
   db2vccmd register
   ```

3. Seleccione **Herramientas —> Personalizar**. Se abrirá el cuaderno Personalizar.
4. Seleccione la pestaña **Macros automáticas y archivos de macro**. Se abrirá la página Macros automáticas y archivos de macro.
5. Seleccione el recuadro **Macro automática de proyectos de IBM DB2**.
6. Pulse **Aceptar**. Se creará una barra de herramientas flotante.

Conceptos relacionados:

- “La macro automática de proyectos de IBM DB2 Universal Database para Microsoft Visual C++” en la página 69

Tareas relacionadas:

- “Activación de la macro automática de herramientas de IBM DB2 Universal Database para Microsoft Visual C++” en la página 74

Consulta relacionada:

- “Terminología de la macro automática de proyectos de IBM DB2 Universal Database para Microsoft Visual C++” en la página 72

Activación de la macro automática de herramientas de IBM DB2 Universal Database para Microsoft Visual C++

La macro automática de herramientas de DB2 es una barra de herramientas que permite ejecutar algunas de las herramientas de desarrollo y administración de DB2 desde dentro del entorno de desarrollo integrado de Visual C++.

Procedimiento:

Para activar la macro automática de herramientas de IBM DB2 Universal Database para Microsoft Visual C++, siga los pasos siguientes:

1. Inicie y detenga Visual C++ al menos una vez con su ID de conexión actual. La primera vez que se ejecuta Visual C++, se crea un perfil para el ID de usuario, y esto es lo que se actualiza mediante el mandato `db2vccmd`. Si no lo ha iniciado una vez e intenta ejecutar `db2vccmd`, se puede encontrar errores como los siguientes:

   ```
   "Registering DB2 Project add-in ...Failed! (rc = 2)"
   ```

2. Registre la macro automática, si aún no la ha hecho, especificando lo siguiente en la línea de mandatos:

   ```
   db2vccmd register
   ```

3. Seleccione **Herramientas —> Personalizar**. Se abrirá el cuaderno Personalizar.
4. Seleccione la pestaña Macros automáticas y archivos de macro.
5. Seleccione el recuadro **Macro automática de herramientas de IBM DB2**.
6. Pulse **Aceptar**. Se creará una barra de herramientas flotante.

Nota: si la barra de herramientas se cierra accidentalmente, puede desactivar y volver a activar la macro automática o utilizar las opciones de personalización estándar de Visual C++ para volver a visualizar la barra de herramientas.

Conceptos relacionados:

- “La macro automática de proyectos de IBM DB2 Universal Database para Microsoft Visual C++” en la página 69

Tareas relacionadas:

- “Activación de la macro automática de proyectos de IBM DB2 Universal Database para Microsoft Visual C++” en la página 73

Consulta relacionada:

- “Terminología de la macro automática de proyectos de IBM DB2 Universal Database para Microsoft Visual C++” en la página 72

Parte 2. SQL incorporado

Capítulo 3. Visión general sobre SQL incorporado

Incorporación de sentencias de SQL en un lenguaje principal

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

Procedimiento:

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

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

Lenguaje	Código fuente de ejemplo

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

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

Tabla 2. Incorporación de sentencias de SQL en un lenguaje principal (continuación)

Lenguaje	Código fuente de ejemplo

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

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

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

- Para los lenguajes C, C++, COBOL o FORTRAN, esta conversión la maneja el precompilador de DB2. El precompilador de DB2 se invoca utilizando el mandato PREP. El precompilador convierte las sentencias de SQL incorporado directamente en llamadas a API de servicios de tiempo de ejecución de DB2.
- Para el lenguaje Java, el conversor SQLj convierte cláusulas SQLj en sentencias JDBC. El conversor SQLj se invoca con el mandato `sqlj`.

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

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

Lenguaje	Código fuente de ejemplo

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

Tabla 3. Incorporación de sentencias de SQL en un lenguaje principal (continuación)

Lenguaje	Código fuente de ejemplo

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

COBOL	`* Consulte la documentación de COBOL para ver normas de comentario`
	`* Aquí sólo se permiten comentarios de COBOL`
	`EXEC SQL`
	`-- Aquí se permiten comentarios de SQL`
	`* o comentarios de COBOL de línea completa`
	`DECLARE C1 CURSOR FOR sname END-EXEC.`
	`* Aquí sólo se permiten comentarios de COBOL`

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

Conceptos relacionados:

- "SQL incorporado en aplicaciones REXX" en la página 372
- "Sentencias de SQL incorporado en C y C++" en la página 182
- "Sentencias de SQL incorporado en COBOL" en la página 237
- "Sentencias de SQL incorporado en FORTRAN" en la página 267
- "Sentencias de SQL incorporado en Java" en la página 306

Creación y preparación del archivo fuente

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

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

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

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

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

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

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

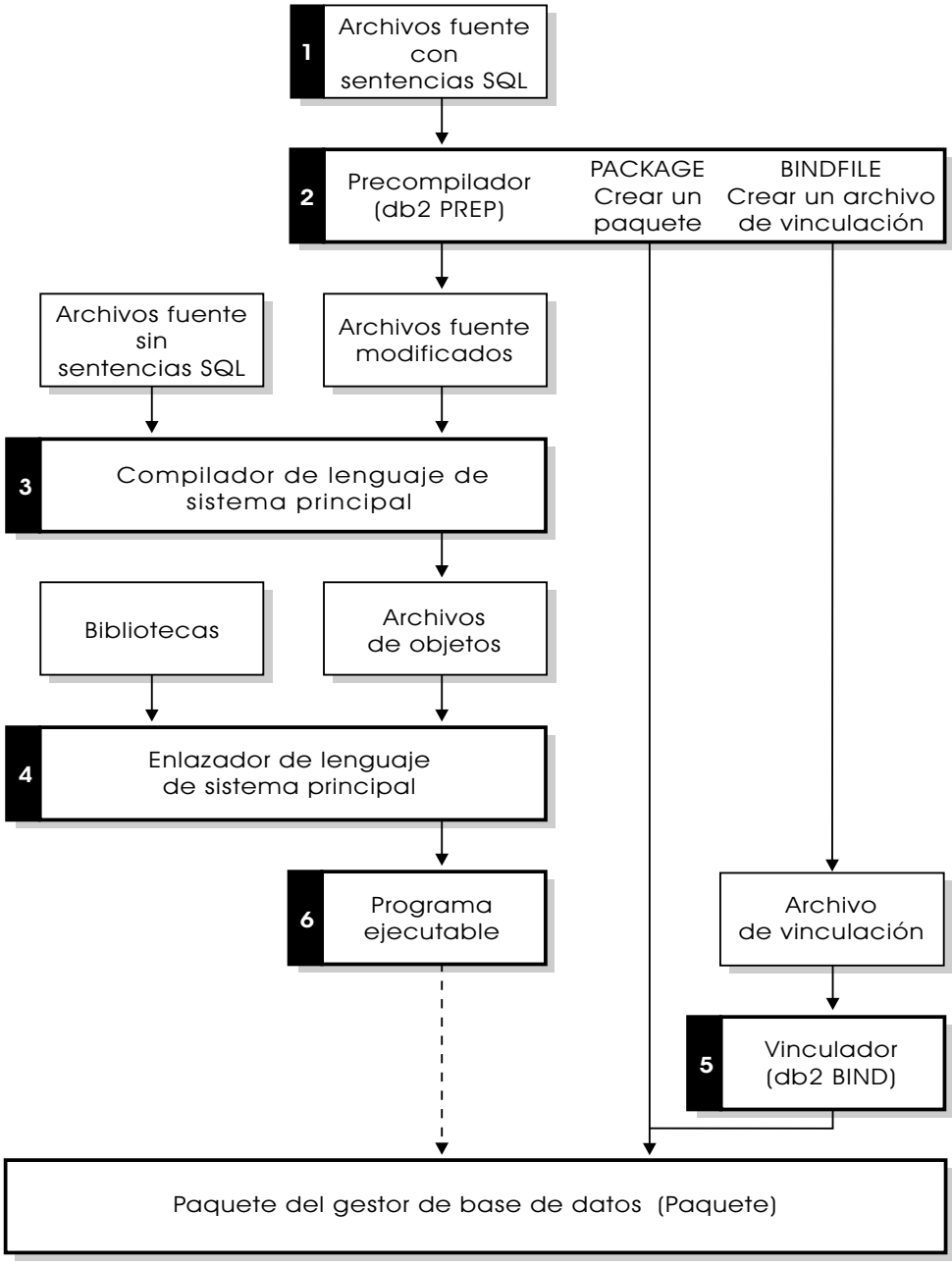


Figura 1. Preparación de programas escritos en lenguajes principal compilados

Conceptos relacionados:

- "Precompilación de archivos fuente que contienen SQL incorporado" en la página 84

- “Requisitos del archivo fuente para aplicaciones de SQL incorporado” en la página 86
- “Compilación y enlace de archivos fuente que contienen SQL incorporado” en la página 88
- “SQL incorporado” en la página 9

Consulta relacionada:

- “Mandato BIND” en el manual *Consulta de mandatos*

Paquetes, vinculación y SQL incorporado

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

Creación de paquetes para SQL incorporado

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

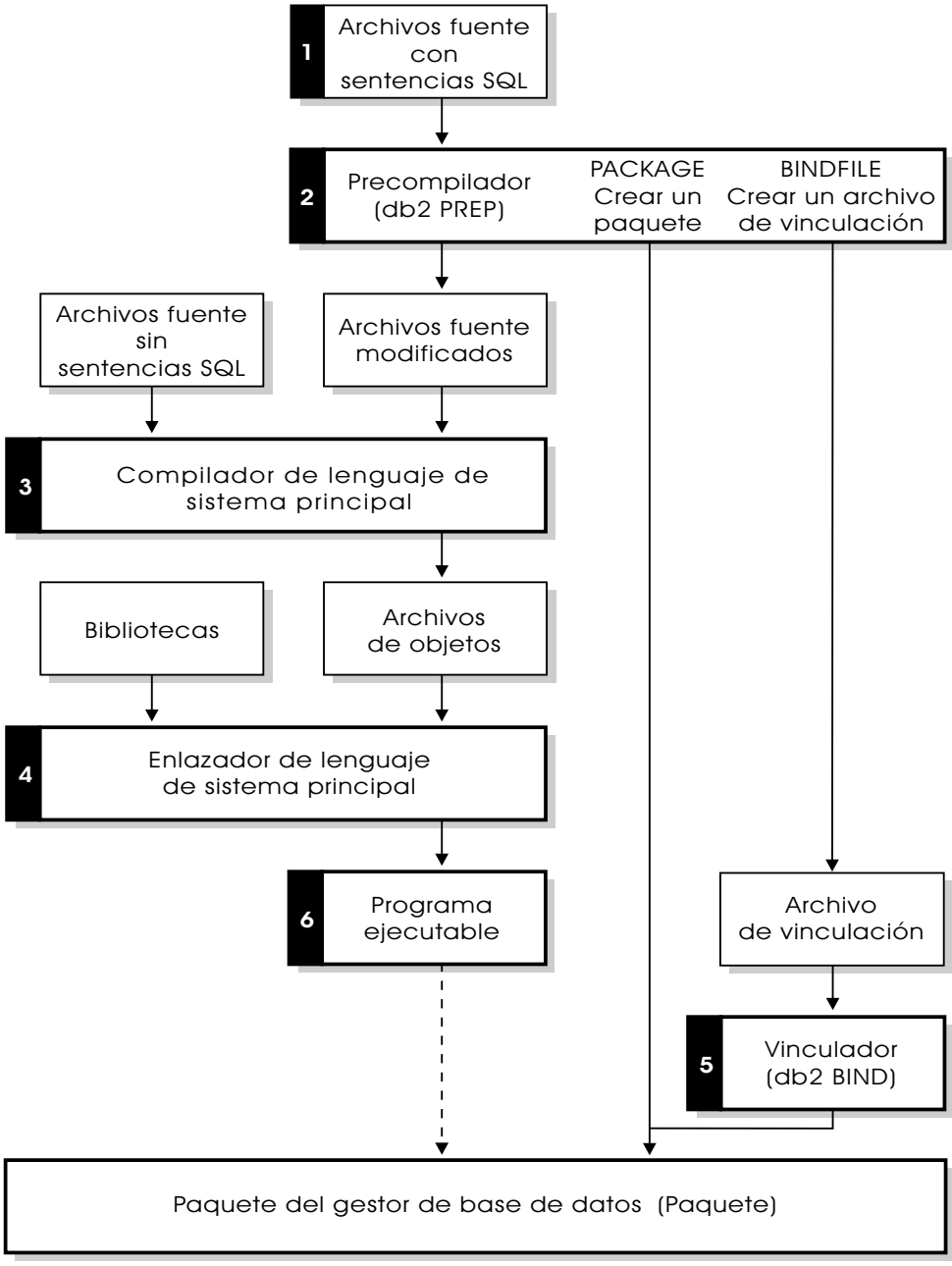


Figura 2. Preparación de programas escritos en lenguajes principales compilados

- Precompilación (paso 2), para convertir las sentencias fuente de SQL incorporado en un formato que el gestor de bases de datos pueda utilizar

- Compilación y enlace (pasos 3 y 4), para crear los módulos de objetos necesarios, y
- Vinculación (paso 5), para crear el paquete que utilizará el gestor de bases de datos cuando se ejecute el programa.

Conceptos relacionados:

- “Precompilación de archivos fuente que contienen SQL incorporado” en la página 84
- “Requisitos del archivo fuente para aplicaciones de SQL incorporado” en la página 86
- “Compilación y enlace de archivos fuente que contienen SQL incorporado” en la página 88
- “Creación de paquetes mediante el mandato BIND” en la página 89
- “Creación de versiones de paquetes” en la página 90
- “Efecto de registros especiales en SQL dinámico vinculado” en la página 91
- “Resolución de nombres de tabla no calificados” en la página 92
- “Consideraciones adicionales cuando se vincula” en la página 93
- “Ventajas de la vinculación diferida” en la página 94
- “Relaciones entre aplicación, archivo de vinculación y paquete” en la página 95
- “Indicaciones horarias generadas por el precompilador” en la página 96
- “Revinculación de paquetes” en la página 98
- “Opciones del conversor SQLj” en la página 312

Consulta relacionada:

- “db2bfd - Mandato Herramienta de descripción de archivo de vinculación” en el manual *Consulta de mandatos*

Precompilación de archivos fuente que contienen SQL incorporado

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

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

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

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

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

El precompilador genera un máximo de cuatro tipos de salida:

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

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

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

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

Archivo de vinculación
Si utiliza la opción BINDFILE, el precompilador crea un

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

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

Archivo de mensajes

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

Conceptos relacionados:

- "Creación de versiones de paquetes" en la página 90

Consulta relacionada:

- "Mandato PRECOMPILE" en el manual *Consulta de mandatos*

Requisitos del archivo fuente para aplicaciones de SQL incorporado

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

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

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

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

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

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

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

Conceptos relacionados:

- “Ventajas de la vinculación diferida” en la página 94

- “Conversión de caracteres entre distintas páginas de códigos” en la página 436
- “Cuándo se produce la conversión de página de códigos” en la página 437
- “Sustitución de caracteres durante conversiones de páginas de códigos” en la página 438
- “Conversiones de páginas de códigos soportadas” en la página 438
- “Factor de expansión de conversión de página de códigos” en la página 440

Consulta relacionada:

- “Mandato PRECOMPILE” en el manual *Consulta de mandatos*

Compilación y enlace de archivos fuente que contienen SQL incorporado

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

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

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

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

Nota: aunque las aplicaciones pueden ser DLL en sistemas operativos Windows, es la aplicación y no el gestor de bases de datos de DB2 la que carga directamente las DLL. En sistemas operativos Windows, el gestor de bases de datos puede cargar DLL. Los procedimientos almacenados se crean normalmente como DLL o bibliotecas compartidas.

Para crear el archivo ejecutable, enlace lo siguiente:

- Módulos de objetos de usuario, generados por el compilador del lenguaje a partir de los archivos fuente modificados y otros archivos que no contienen sentencias de SQL.
- API de biblioteca de lenguaje principal, que se suministran con el compilador del lenguaje.

- La biblioteca del gestor de bases de datos que contiene las API del gestor de bases de datos correspondientes a su entorno operativo. Consulte la documentación sobre programación adecuada para su plataforma operativa para ver el nombre específico de la biblioteca del gestor de bases de datos que necesita para las API del gestor de bases de datos.

Conceptos relacionados:

- "Procedimientos almacenados de DB2" en la página 23

Tareas relacionadas:

- "Creación y ejecución de aplicaciones REXX" en la página 383
- "Creación de applets JDBC" en el manual *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Creación de aplicaciones JDBC" en el manual *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Creación de applets SQLJ" en el manual *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Creación de aplicaciones SQLJ" en el manual *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Creación de aplicaciones C en AIX" en el manual *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Creación de aplicaciones C++ en AIX" en el manual *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Creación de aplicaciones IBM COBOL en AIX" en el manual *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Creación de aplicaciones Micro Focus COBOL en AIX" en el manual *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

Creación de paquetes mediante el mandato BIND

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

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

```
DB2 BIND filename.bnd
```

Se crea un paquete para cada módulo de código fuente precompilado por separado. Si una aplicación tiene cinco archivos fuente, de los cuales tres

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

Consulta relacionada:

- "Mandato BIND" en el manual *Consulta de mandatos*
- "Mandato PRECOMPILE" en el manual *Consulta de mandatos*

Creación de versiones de paquetes

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

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

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

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

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

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

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

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

Conceptos relacionados:

- "Indicaciones horarias generadas por el precompilador" en la página 96

Consulta relacionada:

- "Mandato BIND" en el manual *Consulta de mandatos*
- "Mandato PRECOMPILE" en el manual *Consulta de mandatos*

Efecto de registros especiales en SQL dinámico vinculado

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

- El registro especial CURRENT QUERY OPTIMIZATION determina qué clase de optimización se utiliza.
- El registro especial CURRENT PATH determina la vía de acceso de función utilizada para la resolución de UDF y UDT.
- El registro CURRENT EXPLAIN SNAPSHOT determina qué información de instantánea de Explain se captura.

- El registro CURRENT EXPLAIN MODE determina si la información de tabla de Explain se captura, para cualquier sentencia de SQL dinámico que se pueda elegir. Los valores por omisión correspondientes a estos registros especiales son los mismos que se utilizan para las opciones de vinculación relacionadas.

Consulta relacionada:

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

Resolución de nombres de tabla no calificados

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

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

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

  En el ejemplo anterior, *db_name* es el nombre de la base de datos, *user_name* es el nombre del usuario y *file_name* es el nombre de la aplicación que se va a vincular. Tenga en cuenta que *user_name* y *schema_name* suelen tener el mismo valor. Luego utilice la sentencia SET CURRENT PACKAGESET para especificar qué paquete utilizar y, por lo tanto, qué calificadores se utilizarán. El calificador por omisión es el identificador de autorización que se utiliza al vincular el paquete.
- Cree vistas para cada usuario con el mismo nombre que la tabla de modo que los nombres de tabla no calificados se resuelvan correctamente.
- Cree un alias para que cada usuario apunte a la tabla deseada.

Consulta relacionada:

- "SET CURRENT PACKAGESET sentencia" en el manual *Consulta de SQL, Volumen 2*
- "Mandato BIND" en el manual *Consulta de mandatos*

Consideraciones adicionales cuando se vincula

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

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

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

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

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

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

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

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

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

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

Conceptos relacionados:

- "Binding utilities to the database" en el manual *Administration Guide: Implementation*
- "Conversión de caracteres entre distintas páginas de códigos" en la página 436
- "Sustitución de caracteres durante conversiones de páginas de códigos" en la página 438
- "Factor de expansión de conversión de página de códigos" en la página 440

Consulta relacionada:

- "Mandato BIND" en el manual *Consulta de mandatos*

Ventajas de la vinculación diferida

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

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

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

Consulta relacionada:

- "sqlabndx - Bind" en el manual *Administrative API Reference*

Contenido del archivo de vinculación

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

Consulta relacionada:

- "db2bfd - Mandato Herramienta de descripción de archivo de vinculación" en el manual *Consulta de mandatos*

Relaciones entre aplicación, archivo de vinculación y paquete

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

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

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

Nota: no dé por supuesto que una versión estática de una sentencia de SQL se ejecuta automáticamente más rápido que la misma sentencia procesada de forma dinámica. En algunos casos, SQL estático es más rápido debido al proceso general necesario para preparar la sentencia

dinámica. En otros casos, la misma sentencia preparada de forma dinámica se ejecuta más rápido, porque el optimizador puede utilizar las estadísticas actuales de la base de datos en lugar de las estadísticas de base de datos disponibles en el momento de la vinculación anterior. Observe que si la transacción tarda menos de un par de segundos en completarse, SQL estático suele ser más rápido. Para elegir el método a utilizar, debe realiza prototipos de ambas formas de vinculación.

Conceptos relacionados:

• "SQL dinámico frente a SQL estático" en la página 141

Indicaciones horarias generadas por el precompilador

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

Cuando una aplicación se precompila con la vinculación habilitada, el paquete y el archivo fuente modificado se generan con indicaciones horarias que coinciden. Si existen varias versiones de un paquete (utilizando la opción PRECOMPILE VERSION), cada versión contendrá una indicación horaria asociada. Cuando se ejecuta una aplicación, el nombre del paquete, el creador y la indicación horaria se envían al gestor de bases de datos, el cual comprueba si hay algún paquete cuyo nombre, creador e indicación horaria coinciden con los enviados por la aplicación. Si no existe dicha coincidencia, se devuelve uno de los siguientes códigos de error de SQL a la aplicación:

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

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

encontrado puede proceder del uso de la opción de precompilación o de vinculación ACTION REPLACE REPLVER, como en el siguiente ejemplo:

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

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

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

Un síntoma parecido sucedería en el siguiente ejemplo:

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

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

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

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

Conceptos relacionados:

• “Creación de paquetes mediante el mandato BIND” en la página 89

Revinculación de paquetes

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

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

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

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

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

Conceptos relacionados:

- "Statement dependencies when changing objects" en el manual *Administration Guide: Implementation*

Consulta relacionada:

- "Mandato BIND" en el manual *Consulta de mandatos*
- "Mandato REBIND" en el manual *Consulta de mandatos*

Capítulo 4. Cómo escribir programas de SQL estático

Características de SQL estático y razones para utilizarlo

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

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

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

Cuando se prepara una sentencia de SQL estático, se crea un formato ejecutable de la sentencia y se almacena en el paquete en la base de datos. El formato ejecutable se puede construir en el momento de la precompilación o en un momento de vinculación posterior. En cualquier caso, la preparación se produce *antes* del tiempo de ejecución. Se utiliza la autorización de la persona que vincula la aplicación y la optimización se basa en estadísticas de base de datos y en parámetros de configuración que pueden no estar actualizados cuando se ejecuta la aplicación.

Ventajas del SQL estático

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

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

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

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

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

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

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

Consulta relacionada:

- "EXECUTE sentencia" en el manual *Consulta de SQL, Volumen 2*

Programa de SQL estático de ejemplo

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

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

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

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

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

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

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

- Java (TbRead.sqlj)

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

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

- COBOL (**static.sqb**)

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

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

Conceptos relacionados:

- “Recuperación de datos en programas de SQL estático” en la página 105
- “Recuperación de mensajes de error en una aplicación” en la página 137

Tareas relacionadas:

- “Declaración de variables del lenguaje principal en programas de SQL estático” en la página 108
- “Selección de varias filas utilizando un cursor” en la página 118
- “Configuración de la base de datos sample” en el manual *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

Consulta relacionada:

- "SELECT INTO sentencia" en el manual *Consulta de SQL, Volumen 2*

Ejemplos relacionados:

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

Recuperación de datos en programas de SQL estático

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

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

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

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

Conceptos relacionados:

- "Variables del lenguaje principal en SQL estático" en la página 106
- "Ejemplo de un cursor en un programa de SQL estático" en la página 122

Tareas relacionadas:

- “Declaración de variables del lenguaje principal en programas de SQL estático” en la página 108
- “Cómo hacer referencia a variables del lenguaje principal en programas de SQL estático” en la página 110
- “Inclusión de variables de indicador en programas de SQL estático” en la página 110
- “Selección de varias filas utilizando un cursor” en la página 118
- “Declaración y utilización de cursores en programas de SQL estático” en la página 119

Variables del lenguaje principal en programas de SQL estático

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

Variables del lenguaje principal en SQL estático

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

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

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

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

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

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

- El precompilador desconoce las normas de establecimiento de ámbito de variables del lenguaje principal.

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

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

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


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

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

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

```
}

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

Conceptos relacionados:

- “Variables del lenguaje principal en C y C++” en la página 183
- “Variables del lenguaje principal en COBOL” en la página 240
- “Variables del lenguaje principal en FORTRAN” en la página 269
- “Variables del lenguaje principal en Java” en la página 290
- “Variables del lenguaje principal en REXX” en la página 374

Tareas relacionadas:

- “Declaración de variables del lenguaje principal con el Generador de declaraciones db2dclgn” en la página 38
- “Declaración de variables del lenguaje principal en programas de SQL estático” en la página 108
- “Cómo hacer referencia a variables del lenguaje principal en programas de SQL estático” en la página 110

Declaración de variables del lenguaje principal en programas de SQL estático

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

Procedimiento:

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

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

Lenguaje	Código fuente de ejemplo

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

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

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

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

Tareas relacionadas:

- “Declaración de variables del lenguaje principal con el Generador de declaraciones db2dclgn” en la página 38
- “Cómo hacer referencia a variables del lenguaje principal en programas de SQL estático” en la página 110

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

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

Procedimiento:

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

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

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

Tareas relacionadas:

- “Declaración de variables del lenguaje principal con el Generador de declaraciones db2dclgn” en la página 38
- “Declaración de variables del lenguaje principal en programas de SQL estático” en la página 108

Variables de indicador en programas de SQL estático

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

Inclusión de variables de indicador en programas de SQL estático

Las aplicaciones escritas en lenguajes que no sean Java deben prepararse para recibir valores nulos asociando una *variable de indicador* con cualquier variable del lenguaje principal que pueda recibir un nulo. Las aplicaciones Java comparan el valor de la variable del lenguaje principal con `null` de Java para

determinar si el valor recibido es nulo. El gestor de bases de datos y la aplicación de sistema principal comparten una variable de indicador; por lo tanto, la variable de indicador se debe declarar en la aplicación como una variable del lenguaje principal. Esta variable del lenguaje principal corresponde al tipo SMALLINT de datos de SQL.

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

Procedimiento:

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

Tabla 6. Variables de indicador por lenguaje principal

Lenguaje	Código fuente de ejemplo

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

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

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

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

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

Nota: si el parámetro de configuración de base de datos *dft_sqlmathwarn* tiene el valor 'YES', el valor de *cmind* puede ser -2. Este valor indica un valor NULL causado por la evaluación de una expresión con un error aritmético o por un desbordamiento al intentar convertir el valor de resultado numérico en la variable del lenguaje principal.

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

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

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

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

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

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

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

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

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

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

Tareas relacionadas:

- "Declaración de variables del lenguaje principal con el Generador de declaraciones db2dclgn" en la página 38
- "Declaración de variables del lenguaje principal en programas de SQL estático" en la página 108
- "Cómo hacer referencia a variables del lenguaje principal en programas de SQL estático" en la página 110

Consulta relacionada:

- "Tipos de datos para variables de indicador en programas de SQL estático" en la página 113

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

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

SMALLINT
Entero con signo de 16 bits.

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

BIGINT
Entero con signo de 64 bits.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Notas:

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

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

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

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

Dentro de sentencias de SQL, DB2 proporciona conversiones entre tipos de datos compatibles. Por ejemplo, en la siguiente sentencia SELECT, SALARY y BONUS son columnas de tipo DECIMAL; sin embargo, la compensación total de cada empleado se devuelve como datos de tipo DOUBLE:

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

Observe que la ejecución de la sentencia anterior incluye la conversión entre los tipos de datos DECIMAL y DOUBLE.

Para que los resultados de la consulta resulten más fáciles de leer en pantalla, puede utilizar la siguiente sentencia SELECT:

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

Para convertir datos dentro de la aplicación, póngase en contacto con el proveedor del compilador para obtener rutinas, clases, tipos integrados o API adicionales que den soporte a esta conversión.

Si la página de códigos de la aplicación no coincide con la página de códigos de la base de datos, es posible que los tipos de datos de caracteres también estén sujetos a la conversión de caracteres.

Conceptos relacionados:

- "Consideraciones sobre la conversión de datos" en el manual *Guía de desarrollo de aplicaciones: Programación de aplicaciones de servidor*
- "Conversión de caracteres entre distintas páginas de códigos" en la página 436

Consulta relacionada:

- "CREATE TABLE sentencia" en el manual *Consulta de SQL, Volumen 2*
- "Tipos de datos SQL soportados en C y C++" en la página 218
- "Tipos de datos de SQL soportados en COBOL" en la página 254
- "Tipos de datos SQL soportados en FORTRAN" en la página 276
- "Tipos de datos SQL soportados en Java" en la página 291
- "Tipos de datos SQL soportados en REXX" en la página 381

Ejemplo de variable de indicador en un programa de SQL estático

A continuación se muestran ejemplos de cómo utilizar programas C/C++ de variables de indicador que utilizan SQL estático:

- Ejemplo 1

  El siguiente ejemplo muestra la implantación de variables de indicador en columnas de datos que pueden tener valores null. En este ejemplo, la columna FIRSTNAME no puede contener valores null, pero la columna WORKDEPT sí puede.

```
EXEC SQL BEGIN DECLARE SECTION;
  char wd[3];
  short wd_ind;
  char firstname[13];
EXEC SQL END DECLARE SECTION;

  /* conectar con base de datos de ejemplo */

EXEC SQL SELECT FIRSTNME, WORKDEPT
  INTO :firstname, :wd:wdind
  FROM EMPLOYEE
  WHERE LASTNAME = 'JOHNSON';
```

Puesto que la columna WORKDEPT puede tener un valor null, se debe declarar una variable de indicador como una variable del lenguaje principal antes de que se utilice.

- Ejemplo 2 (**dtlob**)

  El ejemplo **dtlob** tiene una función denominada `BlobFileUse()`. La función `BlobFileUse()` contiene una consulta que lee datos BLOB en un archivo mediante una sentencia SELECT INTO:

```
EXEC SQL BEGIN DECLARE SECTION;
  SQL TYPE IS BLOB_FILE blobFilePhoto;
  char photoFormat[10];
  char empno[7];
  short lobind;
EXEC SQL END DECLARE SECTION;

  /* Conectar con la base de datos de ejemplo */

SELECT picture INTO :blobFilePhoto:lobind
  FROM emp_photo
  WHERE photo_format = :photoFormat AND empno = '000130'
```

  Puesto que la columna BLOBFILEPHOTO puede tener un valor null, se debe declarar una variable de indicador LOBIND como una variable del lenguaje principal antes de que se utilice. El ejemplo **dtlob** muestra cómo trabajar con LOB. Consulte los ejemplos para obtener más información sobre cómo utilizar LOB.

Conceptos relacionados:

- “Programa de SQL estático de ejemplo” en la página 103

Tareas relacionadas:

- “Inclusión de variables de indicador en programas de SQL estático” en la página 110

Consulta relacionada:

- “Tipos de datos para variables de indicador en programas de SQL estático” en la página 113

Ejemplos relacionados:

- “dtlob.out -- HOW TO USE THE LOB DATA TYPE (C)”
- “dtlob.sqc -- How to use the LOB data type (C)”
- “dtlob.out -- HOW TO USE THE LOB DATA TYPE (C++)”
- “dtlob.sqC -- How to use the LOB data type (C++)”

Selección de varias filas mediante un cursor

Las secciones siguientes describen cómo seleccionar filas mediante un cursor. Los programas de ejemplo que muestran cómo declarar un cursor, abrir el cursor, captar filas de la tabla y cerrar el cursor también se describen brevemente.

Selección de varias filas utilizando un cursor

Para permitir que una aplicación recupere un conjunto de filas, SQL utiliza un mecanismo denominado un *cursor*.

Para ayudarle a comprender el concepto de un cursor, supongamos que el gestor de bases de datos crea una *tabla de resultados* que contenga todas las filas recuperadas al ejecutar una sentencia SELECT. Un cursor permite que las filas procedentes de la tabla de resultados estén disponibles para una aplicación, identificando o haciendo referencia a una *fila actual* de esta tabla. Cuando se utiliza un cursor, una aplicación puede recuperar cada fila secuencialmente de la tabla de resultados hasta que se encuentra una condición de fin de datos, es decir, se alcanza la condición NOT FOUND, SQLCODE +100 (SQLSTATE 02000). El conjunto de filas obtenido como resultado de ejecutar la sentencia SELECT puede constar de cero, una o más filas, en función del número de filas que cumplan con la condición de búsqueda.

Procedimiento:

Los pasos a seguir para procesar un cursor son los siguientes:

1. Especifique el cursor utilizando una sentencia DECLARE CURSOR.
2. Realice la consulta y cree la tabla de resultados utilizando la sentencia OPEN.
3. Recupere filas de una en una utilizando la sentencia FETCH.
4. Procese las filas con las sentencias DELETE o UPDATE (si hace falta).
5. Termine el cursor utilizando la sentencia CLOSE.

Una aplicación puede utilizar varios cursores simultáneamente. Cada cursor necesita su propio conjunto de sentencias DECLARE CURSOR, OPEN, CLOSE y FETCH.

Conceptos relacionados:

• “Ejemplo de un cursor en un programa de SQL estático” en la página 122

Declaración y utilización de cursores en programas de SQL estático

Utilice la sentencia DECLARE CURSOR para definir y nombrar el cursor y para identificar el conjunto de filas que se tienen que recuperar mediante una sentencia SELECT.

La aplicación asigna un nombre para el cursor. Se hace referencia a este nombre en siguientes sentencias OPEN, FETCH y CLOSE. La consulta es cualquier sentencia select válida.

Restricciones:

La ubicación de la sentencia DECLARE es arbitraria, pero debe estar antes de la primera vez que se utiliza el cursor.

Procedimiento:

Utilice la sentencia DECLARE para definir el cursor. La tabla siguiente contiene ejemplos correspondientes a los lenguajes principales soportados:

Tabla 7. Declaraciones de cursor por lenguaje principal

Lenguaje	Código fuente de ejemplo

C/C++	`EXEC SQL DECLARE C1 CURSOR FOR`
	`  SELECT PNAME, DEPT FROM STAFF`
	`  WHERE JOB=:host_var;`

JAVA (SQLj)	`#sql iterator cursor1(host_var data type);`
	`#sql cursor1 = { SELECT PNAME, DEPT FROM STAFF`
	`  WHERE JOB=:host_var };`

COBOL	`EXEC SQL DECLARE C1 CURSOR FOR`
	`  SELECT NAME, DEPT FROM STAFF`
	`  WHERE JOB=:host-var END-EXEC.`

FORTRAN	`EXEC SQL DECLARE C1 CURSOR FOR`
	`+  SELECT NAME, DEPT FROM STAFF`
	`+  WHERE JOB=:host_var`

Conceptos relacionados:

- “Consideraciones sobre tipos de cursor y unidad de trabajo” en la página 120

Tareas relacionadas:

- “Selección de varias filas utilizando un cursor” en la página 118

Consulta relacionada:

• “Tipos de cursor” en la página 125

Consideraciones sobre tipos de cursor y unidad de trabajo

Las acciones de una operación COMMIT o ROLLBACK varían para los cursores en función del modo en que éstos se declaren:

Cursores de sólo lectura

Si se determina que un cursor es de sólo lectura y utiliza un nivel de aislamiento de lectura repetitiva, se siguen obteniendo y manteniendo bloqueos de lectura repetitiva en las tablas del sistema que necesita la unidad de trabajo. Por lo tanto, es importante que las aplicaciones emitan periódicamente sentencias COMMIT, incluso para cursores de sólo lectura.

Opción WITH HOLD

Si una aplicación completa una unidad de trabajo emitiendo una sentencia COMMIT, el gestor de bases de datos cierra *todos los cursores abiertos*, excepto aquellos declarados utilizando la opción WITH HOLD.

Un cursor declarado con la opción WITH HOLD mantiene los recursos a los que accede en varias unidades de trabajo. El efecto exacto de declarar un cursor WITH HOLD depende del modo en que finaliza la unidad de trabajo:

- Si la unidad de trabajo finaliza con una sentencia COMMIT, los cursores definidos con WITH HOLD permanecen abiertos (OPEN). El cursor se coloca antes de la siguiente fila lógica de la tabla de resultados. Además, las sentencias preparadas que hacen referencia a cursores OPEN definidos con WITH HOLD se retienen. Sólo las peticiones FETCH y CLOSE asociadas con un determinado cursor son válidas inmediatamente después de la sentencia COMMIT. Las sentencias UPDATE WHERE CURRENT OF y DELETE WHERE CURRENT OF sólo son válidas para filas captadas dentro de la misma unidad de trabajo.

  Nota: si un paquete se revincula durante una unidad de trabajo, todos los cursores retenidos se cierran.
- Si la unidad de trabajo finaliza con una sentencia ROLLBACK, todos los cursores abiertos se cierran, todos los bloqueos adquiridos durante la unidad de trabajo se liberan y todas las sentencias preparadas que dependen del trabajo realizado en dicha unidad se eliminan.

Por ejemplo, supongamos que la tabla TEMPL contiene 1.000 entradas. Desea actualizar la columna salary correspondiente a todos los empleados, y espera emitir una sentencia COMMIT cada vez que actualiza 100 filas.

1. Declare el cursor utilizando la opción WITH HOLD:

```
EXEC SQL DECLARE EMPLUPDT CURSOR WITH HOLD FOR
  SELECT EMPNO, LASTNAME, PHONENO, JOBCODE, SALARY
  FROM TEMPL FOR UPDATE OF SALARY
```

2. Abra el cursor y recupere datos de la tabla de resultados, fila a fila:

```
EXEC SQL OPEN EMPLUPDT
  .
  .
  .

EXEC SQL FETCH EMPLUPDT
  INTO :upd_emp, :upd_lname, :upd_tele, :upd_jobcd, :upd_wage,
```

3. Cuando desee actualizar o suprimir una fila, utilice una sentencia UPDATE o DELETE utilizando la opción WHERE CURRENT OF. Por ejemplo, para actualizar la fila actual, el programa puede emitir:

```
EXEC SQL UPDATE TEMPL SET SALARY = :newsalary
  WHERE CURRENT OF EMPLUPDT
```

4. Después de emitir una sentencia COMMIT, debe emitir una sentencia FETCH antes de poder actualizar otra fila.

Debe incluir código en la aplicación para detectar y manejar un SQLCODE -501 (SQLSTATE 24501), que se puede devolver en una sentencia FETCH o CLOSE si la aplicación:

- Utiliza cursores declarados WITH HOLD.
- Ejecuta más de una unidad de trabajo y deja un cursor WITH HOLD abierto en los límites de la unidad de trabajo (COMMIT WORK).

Si una aplicación invalida su paquete eliminando una tabla de la que depende, el paquete se revincula de forma dinámica. En este caso, se devuelve un SQLCODE -501 (SQLSTATE 24501) para una sentencia FETCH o CLOSE porque el gestor de bases de datos cierra el cursor. El modo de manejar un SQLCODE -501 (SQLSTATE 24501) en esta situación depende de si desea captar filas desde el cursor:

- Si desea captar filas desde el cursor, abra el cursor y luego ejecute la sentencia FETCH. Sin embargo, tenga en cuenta que la sentencia OPEN vuelve a colocar el cursor al principio. La posición anterior retenida en la sentencia COMMIT WORK se pierde.

- Si no desea captar filas desde el cursor, no emita más peticiones SQL contra el cursor.

Opción WITH RELEASE

Cuando una aplicación cierra un cursor utilizando la opción WITH RELEASE, DB2 intenta liberar todos los bloqueos de lectura (READ) que el cursor sigue reteniendo. El cursor sólo continuará reteniendo bloqueos de grabación (WRITE). Si la aplicación cierra el cursor sin utilizar la opción RELEASE, los bloqueos READ y WRITE se liberarán cuando finalice la unidad de trabajo.

Tareas relacionadas:

- "Selección de varias filas utilizando un cursor" en la página 118
- "Declaración y utilización de cursores en programas de SQL estático" en la página 119

Ejemplo de un cursor en un programa de SQL estático

Los ejemplos **tut_read.sqc** en C, **tut_read.sqC/sqx** en C++, **TutRead.sqlj** en SQLj y **cursor.sqb** en COBOL muestran cómo declarar un cursor, abrir el cursor, captar filas de la tabla y cerrar el cursor.

Puesto que REXX no da soporte a SQL estático, no se proporciona ningún ejemplo.

- C/C++

  El ejemplo tut_read muestra una sentencia select básica de una tabla que utiliza un cursor. Por ejemplo:

```
 /* declarar cursor */
    EXEC SQL DECLARE c1 CURSOR FOR
    SELECT deptnumb, deptname FROM org WHERE deptnumb < 40;

/* abrir cursor */
    EXEC SQL OPEN c1 ;

/* captar cursor */
  EXEC SQL FETCH c1 INTO :deptnumb, :deptname;

while (sqlca.sqlcode != 100)
  {
    printf("    %8d %-14s\n", deptnumb, deptname);
    EXEC SQL FETCH c1 INTO :deptnumb, :deptname;
  }

  /* cerrar cursor */
  EXEC SQL CLOSE c1;
```

- Java

  El ejemplo **TutRead** muestra cómo leer datos de una tabla con una sencilla sentencia select utilizando un cursor. Por ejemplo:

```
// definición del cursor
  #sql iterator TutRead_Cursor(int, String);

  // declarar cursor
  TutRead_Cursor cur2;
  #sql cur2 = {SELECT deptnumb, deptname FROM org WHERE deptnumb < 40};

  // captar cursor
  #sql {FETCH :cur2 INTO :deptnumb, :deptname};

  // recuperar y mostrar el resultado de la sentencia SELECT
  while (!cur2.endFetch())
  {
    System.out.println(deptnumb + ", " + deptname);
    #sql {FETCH :cur2 INTO :deptnumb, :deptname};
  }

  // cerrar cursor
  cur2.close();
```

- COBOL

  El ejemplo **cursor** muestra un ejemplo de cómo recuperar datos de una tabla utilizando un cursor con una sentencia de SQL estático. Por ejemplo:

```
* Declarar un cursor
     EXEC SQL DECLARE c1 CURSOR FOR
              SELECT name, dept FROM staff
              WHERE job='Mgr' END-EXEC.

* Abrir el cursor
     EXEC SQL OPEN c1 END-EXEC.

* Captar filas de la tabla 'staff'
     perform Fetch-Loop thru End-Fetch-Loop
        until SQLCODE not equal 0.

* Cerrar el cursor
     EXEC SQL CLOSE c1 END-EXEC.
     move "CLOSE CURSOR" to errloc.
```

Conceptos relacionados:

- “Consideraciones sobre tipos de cursor y unidad de trabajo” en la página 120
- “Recuperación de mensajes de error en una aplicación” en la página 137

Tareas relacionadas:

- “Selección de varias filas utilizando un cursor” en la página 118
- “Declaración y utilización de cursores en programas de SQL estático” en la página 119

Consulta relacionada:

- “Tipos de cursor” en la página 125

Ejemplos relacionados:

- "cursor.sqb -- How to update table data with cursor statically (IBM COBOL)"
- "tut_read.out -- HOW TO READ TABLES (C)"
- "tut_read.sqc -- How to read tables (C)"
- "tut_read.out -- HOW TO READ TABLES (C++)"
- "tut_read.sqC -- How to read tables (C++)"
- "TutRead.out -- HOW TO READ TABLE DATA (SQLJ)"
- "TutRead.sqlj -- Read data in a table (SQLj)"

Manipulación de datos recuperados

Las secciones siguientes describen cómo actualizar y suprimir datos recuperados. Los programas de ejemplo que muestran cómo manipular datos también se describen brevemente.

Actualización y supresión de datos recuperados en programas de SQL estático

Se puede actualizar y suprimir la fila a la que hace referencia un cursor. Para que una fila se pueda actualizar, la consulta correspondiente al cursor no debe ser de sólo lectura.

Procedimiento:

Para actualizar con un cursor, utilice la cláusula WHERE CURRENT OF en una sentencia UPDATE. Utilice la cláusula FOR UPDATE para indicar al sistema que desea actualizar algunas columnas de la tabla de resultados. Puede especificar una columna en la cláusula FOR UPDATE sin que esté en fullselect; por lo tanto, puede actualizar columnas que no recupera explícitamente el cursor. Si la cláusula FOR UPDATE se especifica sin nombres de columna, se considera que todas las columnas de la tabla o vista identificada en la primera cláusula FROM de fullselect externo se pueden actualizar. No nombre más columnas de las que necesita en la cláusula FOR UPDATE. En algunos casos, si se nombran columnas adicionales en la cláusula FOR UPDATE, es posible que DB2 sea menos eficiente al acceder a los datos.

La supresión con un cursor se realiza utilizando la cláusula WHERE CURRENT OF en una sentencia DELETE. En general, la cláusula FOR UPDATE no se necesita para la supresión de la fila actual de un cursor. La única excepción se produce cuando se utiliza SQL dinámico para la sentencia SELECT o la sentencia DELETE en una aplicación que se ha precompilado con el valor SAA1 para LANGLEVEL y se ha vinculado con BLOCKING ALL. En este caso, se necesita una cláusula FOR UPDATE en la sentencia SELECT.

La sentencia DELETE hace que la fila a la que hace referencia el cursor se suprima. Esta supresión deja el cursor colocado antes de la *siguiente* fila y se debe emitir una sentencia FETCH antes de que se puedan realizar operaciones WHERE CURRENT OF adicionales contra el cursor.

Conceptos relacionados:

- "Consultas" en el manual *Consulta de SQL, Volumen 1*

Consulta relacionada:

- "Mandato PRECOMPILE" en el manual *Consulta de mandatos*

Tipos de cursor

Los cursores se dividen en tres categorías:

Sólo lectura
Las filas del cursor sólo se pueden leer, no actualizar. Los cursores de sólo lectura se utilizan cuando una aplicación sólo va a leer datos, no a modificarlos. Un cursor se considera de sólo lectura si se basa en una sentencia select de sólo lectura. Consulte la descripción sobre cómo actualizar y recuperar datos correspondientes a normas para sentencias select que definen tablas de resultados no actualizables.

Puede haber ventajas en cuanto a rendimiento para cursores de sólo lectura.

Actualizables
Las filas del cursor se pueden actualizar. Los cursores actualizables se utilizan cuando una aplicación modifica datos a medida que se captan las filas en el cursor. La consulta especificada sólo puede hacer referencia a una tabla o vista. La consulta también debe incluir la cláusula FOR UPDATE, nombrando cada columna que se va a actualizar (a no ser que se utilice la opción de precompilación LANGLEVEL MIA).

Ambiguos
No se puede determinar si el cursor es actualizable o de sólo lectura a partir de su definición o contexto. Esta situación puede producirse cuando se encuentra una sentencia de SQL dinámico que se podría utilizar para cambiar un cursor que de otro modo se consideraría de sólo lectura.

Un cursor ambiguo se trata como de sólo lectura si se especifica la opción BLOCKING ALL al precompilar o vincular. De lo contrario, el cursor se considera actualizable.

Nota: los cursores que se procesan de forma dinámica siempre son ambiguos.

Conceptos relacionados:

- “Modalidades de cursor soportadas en IBM OLE DB Provider” en la página 396

Tareas relacionadas:

- “Actualización y supresión de datos recuperados en programas de SQL estático” en la página 124

Ejemplo de captación en un programa de SQL estático

En el siguiente ejemplo se efectúa una selección en una tabla utilizando un cursor, se abre el cursor y se captan filas de la tabla. Para cada fila captada, el programa decide, según criterios sencillos, si la fila se debe suprimir o actualizar.

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

- C/C++ (**tut_mod.sqc/tut_mod.sqC**)

  El siguiente ejemplo procede del ejemplo **tut_mod**. En este ejemplo se efectúa una selección en una tabla utilizando un cursor, se abre el cursor, se captan, se actualizan o se suprimen filas de la tabla y luego se cierra el cursor.

```
EXEC SQL DECLARE c1 CURSOR FOR SELECT * FROM staff WHERE id >= 310;
  EXEC SQL OPEN c1;
  EXEC SQL FETCH c1 INTO :id, :name, :dept, :job:jobInd, :years:yearsInd, :salary,
     :comm:commInd;
```

  El ejemplo **tbmod** es una versión más larga del ejemplo **tut_mod** y muestra casi todos los casos posibles de modificación de datos de la tabla.

- Java (**TutMod.sqlj**)

  El siguiente ejemplo procede del ejemplo **TutMod**. En este ejemplo se efectúa una selección en una tabla utilizando un cursor, se abre el cursor, se captan, se actualizan o se suprimen filas de la tabla y luego cierra el cursor.

```
#sql cur = {SELECT * FROM staff WHERE id >= 310};
#sql {FETCH :cur INTO :id, :name, :dept, :job, :years, :salary, :comm};
```

  El ejemplo **TbMod** es una versión más larga del ejemplo **TutMod** y muestra casi todos los casos posibles de modificación de datos de la tabla.

- COBOL (**openftch.sqb**)

  El siguiente ejemplo procede del ejemplo **openftch**. En este ejemplo se efectúa una selección en una tabla utilizando un cursor, se abre el cursor y se captan filas de la tabla.

```
 EXEC SQL DECLARE c1 CURSOR FOR
   SELECT name, dept FROM staff
   WHERE job='Mgr'
```

```
    FOR UPDATE OF job END-EXEC.

  EXEC SQL OPEN c1 END-EXEC


 * llamar a FETCH y al bucle UPDATE/DELETE.
   perform Fetch-Loop thru End-Fetch-Loop
   until SQLCODE not equal 0.

  EXEC SQL CLOSE c1 END-EXEC.
```

Conceptos relacionados:

- “Recuperación de mensajes de error en una aplicación” en la página 137

Ejemplos relacionados:

- “openftch.sqb -- How to modify table data using cursor statically (IBM COBOL)”
- “tbmod.sqc -- How to modify table data (C)”
- “tut_mod.out -- HOW TO MODIFY TABLE DATA (C)”
- “tut_mod.sqc -- How to modify table data (C)”
- “tbmod.sqC -- How to modify table data (C++)”
- “tut_mod.out -- HOW TO MODIFY TABLE DATA (C++)”
- “tut_mod.sqC -- How to modify table data (C++)”
- “TbMod.sqlj -- How to modify table data (SQLj)”
- “TutMod.out -- HOW TO MODIFY TABLE DATA (SQLJ)”
- “TutMod.sqlj -- Modify data in a table (SQLj)”

Desplazamiento por datos recuperados y manipulación de los mismos

Las secciones siguientes describen cómo desplazarse por datos recuperados. Los programas de ejemplo que muestran cómo manipular datos también se describen brevemente.

Desplazamiento por datos recuperados previamente

Cuando una aplicación recupera datos de la base de datos, la sentencia FETCH le permite desplazarse hacia adelante por los datos; sin embargo, el gestor de bases de datos no tiene ninguna sentencia de SQL incorporado que le permita desplazarse hacia atrás por los datos (equivalente a una operación FETCH hacia atrás). Sin embargo, las CLI de DB2 y Java dan soporte a una operación FETCH hacia atrás por cursores desplazables de sólo lectura.

Procedimiento:

Para aplicaciones de SQL incorporado, puede utilizar las siguientes técnicas para desplazarse por los datos que se han recuperado:

- Conserve una copia de los datos que se han captado y desplácese por los mismos mediante alguna técnica de programación.
- Utilice SQL para recuperar de nuevo los datos, normalmente mediante una segunda sentencia SELECT.

Conceptos relacionados:

- "Especificación JDBC" en la página 295

Tareas relacionadas:

- "Cómo conservar una copia de los datos" en la página 128
- "Recuperación de datos por segunda vez" en la página 129

Consulta relacionada:

- "SQLFetchScroll Function (CLI) - Fetch Rowset and Return Data for All Bound Columns" en el manual *CLI Guide and Reference, Volume 2*
- "Cursor Positioning Rules for SQLFetchScroll() (CLI)" en el manual *CLI Guide and Reference, Volume 2*

Cómo conservar una copia de los datos

En algunas situaciones, puede resultar útil mantener una copia de los datos que capta la aplicación.

Procedimiento:

Para conservar una copia de los datos, la aplicación puede hacer lo siguiente:

- Guardar los datos captados en almacenamiento virtual.
- Grabar los datos en un archivo temporal (si los datos no caben en almacenamiento virtual). Un efecto de este enfoque es que un usuario que se desplace hacia atrás siempre ve exactamente los mismos datos que se han captado, incluso si mientras tanto una transacción ha modificado los datos de la base de datos.
- Utilizando un nivel de aislamiento de lectura repetida, los datos que recupera de una transacción se pueden volver a recuperar cerrando y abriendo un cursor. Otras aplicaciones no pueden actualizar los datos del conjunto de resultados. Los niveles de aislamiento y el bloqueo pueden afectar al modo en que los usuarios actualizan datos.

Conceptos relacionados:

- "Diferencias en el orden de filas entre la primera y la segunda tabla de resultados" en la página 130

Tareas relacionadas:

- “Recuperación de datos por segunda vez” en la página 129

Recuperación de datos por segunda vez

La técnica que utilice para recuperar datos por segunda vez dependerá del orden en el que desee volver a ver los datos.

Procedimiento:

Puede recuperar datos por segunda vez mediante cualquiera de los métodos siguientes:

- Recuperar datos desde el principio

  Para volver a recuperar los datos desde el principio de la tabla de resultados, cierre el cursor activo y vuélvalo a abrir. Esta acción coloca el cursor al principio de la tabla de resultados. Pero, a no ser que la aplicación retenga bloqueos en la tabla, es posible que otros la hayan modificado, de modo que puede que la que era la primera fila de la tabla de resultados ya no lo sea.

- Recuperar datos desde el medio

  Para recuperar datos por segunda vez desde algún punto del medio de la tabla de resultados, ejecute una segunda sentencia SELECT y declare un segundo cursor en la sentencia. Por ejemplo, supongamos que la primera sentencia SELECT era la siguiente:

  ```
  SELECT * FROM DEPARTMENT
    WHERE LOCATION = 'CALIFORNIA'
    ORDER BY DEPTNO
  ```

  Ahora, supongamos que desea volver a las filas que empiezan con `DEPTNO = 'M95'` y captar de forma secuencial desde dicho punto. Codifique lo siguiente:

  ```
  SELECT * FROM DEPARTMENT
    WHERE LOCATION = 'CALIFORNIA'
    AND DEPTNO >= 'M95'
    ORDER BY DEPTNO
  ```

  Esta sentencia coloca el cursor donde desea.

- Recuperar datos en orden inverso

  El valor por omisión consiste en ordenar las filas en orden ascendente. Si sólo hay una fila para cada valor de `DEPTNO`, la siguiente sentencia especifica un orden ascendente exclusivo de filas:

  ```
  SELECT * FROM DEPARTMENT
    WHERE LOCATION = 'CALIFORNIA'
    ORDER BY DEPTNO
  ```

Para recuperar las mismas filas en orden inverso, especifique que el orden debe ser descendente, como en la siguiente sentencia:

```
SELECT * FROM DEPARTMENT
  WHERE LOCATION = 'CALIFORNIA'
  ORDER BY DEPTNO DESC
```

Un cursor de la segunda sentencia recupera filas exactamente en el orden inverso al de un cursor de la primera sentencia. El orden de recuperación sólo se garantiza si la primera sentencia especifica una secuencia de orden exclusiva.

Para recuperar filas en orden inverso, puede resultar útil tener dos índices en la columna DEPTNO, uno en orden ascendente y el otro en orden descendente.

Conceptos relacionados:

- "Diferencias en el orden de filas entre la primera y la segunda tabla de resultados" en la página 130

Diferencias en el orden de filas entre la primera y la segunda tabla de resultados

Puede que las filas de la segunda tabla de resultados no se visualicen en el mismo orden que en la primera. El gestor de bases de datos no tiene en cuenta el orden de las filas como algo significativo a no ser que la sentencia SELECT utilice ORDER BY. Por lo tanto, si hay varias filas con el mismo valor DEPTNO, puede que la segunda sentencia SELECT las recupere en un orden distinto al de la primera. La única garantía es que todas estarán en orden por número de departamento, tal como solicita la cláusula ORDER BY DEPTNO.

La diferencia en el orden se puede producir aunque ejecute la misma sentencia de SQL, con las mismas variables del lenguaje principal, por segunda vez. Por ejemplo, las estadísticas del catálogo se podrían haber actualizado entre ejecuciones, o se podrían hacer creado o eliminado índices. Entonces podría ejecutar de nuevo la sentencia SELECT.

Es más probable que el orden cambie si la segunda sentencia SELECT tiene un predicado que la primera no tenía; el gestor de bases de datos podría elegir utilizar un índice en el nuevo predicado. Por ejemplo, podría elegir un índice en LOCATION para la primera sentencia de nuestro ejemplo y un índice en DEPTNO para la segunda. Puesto que las filas se captan en orden por la clave de índice, el segundo orden no es necesariamente igual al primero.

De nuevo, al ejecutar dos sentencias SELECT parecidas se puede producir un orden diferente de filas, aunque no se haya producido ningún cambio en las estadísticas ni se haya creado ni eliminado ningún índice. En el ejemplo, si

hay varios valores diferentes de LOCATION, el gestor de bases de datos podría elegir un índice en LOCATION para ambas sentencias. Si se cambia el valor de DEPTNO en la segunda sentencia por lo siguiente, el gestor de bases de datos podría elegir un índice en DEPTNO:

```
SELECT * FROM DEPARTMENT
  WHERE LOCATION = 'CALIFORNIA'
  AND DEPTNO >= 'Z98'
  ORDER BY DEPTNO
```

Debido a la sutil relación entre el formato de una sentencia de SQL y los valores de dicha sentencia, nunca dé por supuesto que dos sentencias de SQL diferentes vayan a devolver filas en el mismo orden, a no ser que el orden se determine de forma exclusiva mediante una cláusula ORDER BY.

Tareas relacionadas:

- "Recuperación de datos por segunda vez" en la página 129

Colocación de un cursor al final de una tabla

Si tiene que colocar el cursor al final de una tabla, puede utilizar una sentencia de SQL para colocarlo.

Procedimiento:

Utilice cualquiera de los ejemplos siguientes como método para colocar un cursor:

- El gestor de bases de datos no garantiza un orden en los datos almacenados en una tabla; por lo tanto, el final de una tabla no está definido. Sin embargo, el orden se define en el resultado de una sentencia de SQL:

  ```
  SELECT * FROM DEPARTMENT
    ORDER BY DEPTNO DESC
  ```

- La sentencia siguiente coloca el cursor en la fila que tiene el valor de DEPTNO más alto:

  ```
  SELECT * FROM DEPARTMENT
    WHERE DEPTNO =
    (SELECT MAX(DEPTNO) FROM DEPARTMENT)
  ```

Sin embargo, observe que, si hay varias filas con el mismo valor, el cursor se coloca en la primera de ellas.

Actualización de datos recuperados previamente

Para desplazarse hacia atrás y actualizar datos recuperados previamente, puede utilizar una combinación de las técnicas que se utilizan para desplazarse a través de datos recuperados previamente y para actualizar datos recuperados.

Procedimiento:

Para actualizar datos recuperados previamente, puede hacer una de estas dos cosas:

- Si tiene un segundo cursor en los datos que se tienen que actualizar y la sentencia SELECT no utiliza ninguno de los elementos restringidos, puede utilizar la sentencia UPDATE controlada por el cursor. Nombre el segundo cursor en la cláusula WHERE CURRENT OF.
- En otros casos, utilice UPDATE con una cláusula WHERE que nombre todos los valores de la fila o especifique la clave primaria de la tabla. Puede ejecutar una sentencia varias veces con distintos valores de las variables.

Tareas relacionadas:

- "Actualización y supresión de datos recuperados en programas de SQL estático" en la página 124
- "Desplazamiento por datos recuperados previamente" en la página 127

Ejemplo de inserción, actualización y supresión en un programa de SQL estático

Los siguientes ejemplos muestran cómo insertar, actualizar y suprimir datos mediante SQL estático.

- C/C++ (**tut_mod.sqc/tut_mod.sqC**)

  Los tres ejemplos siguientes proceden del ejemplo **tut_mod**. Consulte este ejemplo para ver un programa completo que muestra cómo modificar datos de tablas en C o C++.

  El siguiente ejemplo muestra cómo insertar datos de tablas:

```
EXEC SQL INSERT INTO staff(id, name, dept, job, salary)
  VALUES(380, 'Pearce', 38, 'Clerk', 13217.50),
        (390, 'Hachey', 38, 'Mgr', 21270.00),
        (400, 'Wagland', 38, 'Clerk', 14575.00);
```

  El siguiente ejemplo muestra cómo actualizar datos de tablas:

```
EXEC SQL UPDATE staff
  SET salary = salary + 10000
  WHERE id >= 310 AND dept = 84;
```

  El siguiente ejemplo muestra cómo suprimir datos de una tabla:

```
EXEC SQL DELETE
  FROM staff
  WHERE id >= 310 AND salary > 20000;
```

- Java (**TutMod.sqlj**)

  Los tres ejemplos siguientes proceden del ejemplo **TutMod**. Consulte este ejemplo para ver un programa completo que muestra cómo modificar datos de tablas en SQLj.

El siguiente ejemplo muestra cómo insertar datos de tablas:

```
#sql {INSERT INTO staff(id, name, dept, job, salary)
        VALUES(380, 'Pearce', 38, 'Clerk', 13217.50),
              (390, 'Hachey', 38, 'Mgr', 21270.00),
              (400, 'Wagland', 38, 'Clerk', 14575.00)};
```

El siguiente ejemplo muestra cómo actualizar datos de tablas:

```
#sql {UPDATE staff
        SET salary = salary + 1000
        WHERE id >= 310 AND dept = 84};
```

El siguiente ejemplo muestra cómo suprimir datos de una tabla:

```
#sql {DELETE FROM staff
        WHERE id >= 310 AND salary > 20000};
```

- COBOL (**updat.sqb**)

  Los tres ejemplos siguientes proceden del ejemplo **updat**. Consulte este ejemplo para ver un programa completo que muestra cómo modificar datos de tablas en COBOL.

  El siguiente ejemplo muestra cómo insertar datos de tablas:

```
EXEC SQL INSERT INTO staff
    VALUES (999, 'Testing', 99, :job-update, 0, 0, 0)
    END-EXEC.
```

  El siguiente ejemplo muestra cómo actualizar datos de tablas:

```
EXEC SQL UPDATE staff
    SET job=:job-update
    WHERE job='Mgr'
    END-EXEC.
```

  El siguiente ejemplo muestra cómo suprimir datos de una tabla:

```
EXEC SQL DELETE
    FROM staff
    WHERE job=:job-update
    END-EXEC.
```

Conceptos relacionados:

- “Recuperación de mensajes de error en una aplicación” en la página 137

Ejemplos relacionados:

- “tbinfo.out -- HOW TO GET INFORMATION AT THE TABLE LEVEL (C++)”
- “tbmod.out -- HOW TO MODIFY TABLE DATA (C++)”
- “tbmod.sqC -- How to modify table data (C++)”
- “tut_mod.out -- HOW TO MODIFY TABLE DATA (C++)”
- “tut_mod.sqC -- How to modify table data (C++)”

- “tbmod.out -- HOW TO MODIFY TABLE DATA (C)”
- “tbmod.sqc -- How to modify table data (C)”
- “tut_mod.out -- HOW TO MODIFY TABLE DATA (C)”
- “tut_mod.sqc -- How to modify table data (C)”
- “TbMod.out -- HOW TO MODIFY TABLE DATA (SQLJ)”
- “TbMod.sqlj -- How to modify table data (SQLj)”
- “TutMod.out -- HOW TO MODIFY TABLE DATA (SQLJ)”
- “TutMod.sqlj -- Modify data in a table (SQLj)”

Información de diagnóstico

Las secciones siguientes describen la información de diagnóstico disponible para un programa de SQL estático, como códigos de retorno, y cómo debe la aplicación recuperar mensajes de error.

Códigos de retorno

La mayoría de las API del gestor de bases de datos devuelven un código de retorno cero cuando se ejecutan satisfactoriamente. En general, un código de retorno distinto de cero indica que el mecanismo de manejo de errores secundarios, la estructura SQLCA, puede estar dañado. En este caso, la API llamada no se ejecuta. Una posible razón de que se dañe la estructura SQLCA es que se haya pasado una dirección no válida para la estructura.

Consulta relacionada:

- “SQLCA” en el manual *Administrative API Reference*

Información de error en los campos SQLCODE, SQLSTATE y SQLWARN

La información de error se devuelve en los campos SQLCODE y SQLSTATE de la estructura SQLCA, que se actualiza tras cada sentencia de SQL ejecutable y tras la mayoría de las llamadas a API del gestor de bases de datos.

Un archivo fuente que contiene sentencias de SQL ejecutables puede proporcionar al menos una estructura SQLCA con el nombre `sqlca`. La estructura SQLCA se define en el archivo include SQLCA. Los archivos fuente sin sentencias de SQL incorporado, pero que llaman a las API del gestor de bases de datos, también pueden proporcionar una o más estructuras SQLCA, pero sus nombres son arbitrarios.

Si la aplicación cumple con el estándar FIPS 127-2, puede declarar SQLSTATE y SQLCODE como variables del lenguaje principal para aplicaciones C, C++, COBOL y FORTRAN, en lugar de utilizar la estructura SQLCA.

Un valor SQLCODE igual a 0 significa una ejecución satisfactoria (con posibles condiciones de aviso SQLWARN). Un valor positivo significa que la sentencia se ha ejecutado satisfactoriamente pero con un aviso, como el truncamiento de una variable del lenguaje principal. Un valor negativo significa que se ha producido una condición de error.

Un campo adicional, SQLSTATE, contiene un código de error estandarizado coherente con otros productos de bases de datos de IBM y con otros gestores de bases de datos que cumplen con SQL92. En la práctica, debe utilizar SQLSTATE cuando le preocupe la portabilidad, puesto que los SQLSTATE son comunes entre varios gestores de bases de datos.

El campo SQLWARN contiene una matriz de indicadores de aviso, incluso si SQLCODE es cero. El primer elemento de la matriz SQLWARN, SQLWARN0, contiene un blanco si todos los demás elementos están en blanco. SQLWARN0 contiene una `W` si al menos uno de los otros elementos contiene un carácter de aviso.

Nota: si desea desarrollar aplicaciones que acceden a varios servidores RDBMS de IBM, debe:

- Si es posible, hacer que las aplicaciones comprueben SQLSTATE en lugar de SQLCODE.
- Si la aplicación va a utilizar DB2 Connect, considerar la posibilidad de utilizar el recurso de correlación que proporciona DB2 Connect para correlacionar conversiones de SQLCODE entre bases de datos distintas.

Conceptos relacionados:

- “Variables SQLSTATE y SQLCODE en C y C++” en la página 224
- “Variables SQLSTATE y SQLCODE en COBOL” en la página 257
- “Variables SQLSTATE y SQLCODE en FORTRAN” en la página 279
- “Valores de SQLSTATE y SQLCODE en Java” en la página 335
- “Variables SQLSTATE y SQLCODE en Perl” en la página 366

Consulta relacionada:

- “SQLCA” en el manual *Administrative API Reference*

Truncamiento de símbolos en la estructura SQLCA

Puesto que los símbolos se pueden truncar en la estructura SQLCA, no debe utilizar la información de símbolos cuando realice diagnósticos. Aunque puede definir nombres de tabla y de columna con longitudes de hasta 128

bytes, los símbolos de SQLCA se truncarán a 17 bytes más un terminador de truncamiento (>). La lógica de la aplicación no debería depender de los valores reales del campo `sqlerrmc`.

Consulta relacionada:

- "SQLCA" en el manual *Administrative API Reference*

Consideraciones sobre el manejador de excepciones, señales e interrupciones

Un manejador de excepciones, señales o interrupciones es una rutina que adquiere el control cuando se produce una excepción, señal o interrupción. El tipo de manejador aplicable lo determina el entorno operativo, tal como se muestra a continuación:

Sistemas operativos Windows
: Al pulsar `Control-C` o `Control-Inter` se genera una interrupción.

Sistemas basados en UNIX
: Generalmente, al pulsar `Control-C` se genera una señal de interrupción SIGINT. Observe que los teclados se pueden redefinir fácilmente de modo que se pueda generar una señal SIGINT pulsando otra secuencia de teclas en la máquina.

No coloque sentencias de SQL (que no sean COMMIT o ROLLBACK) en manejadores de excepciones, señales e interrupciones. Con estos tipos de condiciones de error, generalmente deseará llevar a cabo una operación ROLLBACK para evitar el riesgo de tener datos incoherentes.

Tenga en cuenta que debe tener cuidado cuando codifique COMMIT y ROLLBACK en manejadores de excepciones/señales/interrupciones. Si llama a cualquiera de estas sentencias por ellas mismas, la operación COMMIT o ROLLBACK no se ejecuta hasta que se completa la sentencia de SQL actual, si se está ejecutando alguna. Este no es el comportamiento deseado de un manejador `Control-C`.

La solución consiste en llamar a la API INTERRUPT (`sqleintr`/`sqlgintr`) antes de emitir una operación ROLLBACK. Esta API interrumpe la consulta de SQL actual (si la aplicación está ejecutando alguna) y deja que ROLLBACK comience de inmediato. Si va a realizar una operación COMMIT en lugar de ROLLBACK, no desea interrumpir el mandato actual.

Cuando se utiliza APPC para acceder a un servidor de base de datos remoto (DB2 para AIX o el sistema de bases de datos de sistema principal que utiliza DB2 Connect), es posible que la aplicación reciba una señal SIGUSR1. SNA Services/6000 genera esta señal cuando se produce un error no recuperable y

la conexión SNA se detiene. Es posible que desee instalar un manejador de señales en la aplicación para que maneje SIGUSR1.

Consulte la documentación de su plataforma para ver detalles específicos sobre consideraciones sobre distintos manejadores.

Conceptos relacionados:

- "Proceso de peticiones de interrupción" en la página 532

Consideraciones sobre rutinas de lista de salida

No utilice SQL ni llamadas a API de DB2 en rutinas de lista de salida. Tenga en cuenta que no puede desconectarse de una base de datos en una rutina de salida.

Recuperación de mensajes de error en una aplicación

En función del lenguaje en el que esté escrita la aplicación, debe utilizar un método u otro para recuperar información de error:

- Las aplicaciones C, C++ y COBOL pueden utilizar la API GET ERROR MESSAGE para obtener la información correspondiente relacionada con el SQLCA que se ha pasado.
- Las aplicaciones JDBC y SQLj emiten una SQLException cuando se produce un error durante el proceso de SQL. Las aplicaciones pueden obtener y visualizar una SQLException con el siguiente código:

```
try {
   Statement stmt = connection.createStatement();
   int rowsDeleted = stmt.executeUpdate(
      "DELETE FROM employee WHERE empno = '000010'");
   System.out.println( rowsDeleted + " rows were deleted");
}

catch (SQLException sqle) {
   System.out.println(sqle);
}
```

- Las aplicaciones REXX utilizan el procedimiento CHECKERR.

Conceptos relacionados:

- "Variables SQLSTATE y SQLCODE en C y C++" en la página 224
- "Variables SQLSTATE y SQLCODE en COBOL" en la página 257
- "Variables SQLSTATE y SQLCODE en FORTRAN" en la página 279
- "Valores de SQLSTATE y SQLCODE en Java" en la página 335
- "Variables SQLSTATE y SQLCODE en Perl" en la página 366

Consulta relacionada:

- "sqlaintp - Get Error Message" en el manual *Administrative API Reference*

Capítulo 5. Cómo escribir programas de SQL dinámico

Características y razones para utilizar SQL dinámico

Las secciones siguientes describen las razones para utilizar SQL dinámico, en comparación con SQL estático.

Razones para utilizar SQL dinámico

Es posible que desee utilizar SQL dinámico si:

- Necesita que toda la sentencia de SQL, o parte de la misma, se genere durante la ejecución de la aplicación.
- Los objetos a los que hace referencia la sentencia de SQL no existen en el momento de la precompilación.
- Desea que la sentencia siempre utilice la vía de acceso óptima, según las estadísticas actuales de la base de datos.

- Desea modificar el entorno de compilación de la sentencia, es decir, experimentar con los registros especiales.

Conceptos relacionados:

- “Sentencias de soporte de SQL dinámico” en la página 140
- “SQL dinámico frente a SQL estático” en la página 141

Sentencias de soporte de SQL dinámico

Las sentencias de soporte de SQL dinámico aceptan una variable del lenguaje principal de serie de caracteres y un nombre de sentencia como argumentos. La variable del lenguaje principal contiene la sentencia de SQL que se va a procesar de forma dinámica en formato de texto. El texto de la sentencia no se procesa cuando se precompila una aplicación. De hecho, el texto de la sentencia no tiene que existir en el momento en que se precompila la aplicación. En su lugar, la sentencia de SQL se trata como una variable del lenguaje principal con fines de precompilación y se hace referencia a la variable durante la ejecución de la aplicación. Estas sentencias de SQL se denominan SQL *dinámico*.

Se necesitan sentencias de soporte de SQL dinámico para transformar la variable del lenguaje principal que contiene texto de SQL en un formato ejecutable y trabajar en el mismo haciendo referencia al nombre de la sentencia. Estas sentencias son:

EXECUTE IMMEDIATE
: Prepara y ejecuta una sentencia que no utiliza ninguna variable del lenguaje principal. Todas las sentencias EXECUTE IMMEDIATE de una aplicación se colocan en antememoria en el mismo lugar en el tiempo de ejecución, de modo que sólo se conoce la última sentencia. Utilice esta sentencia como alternativa a las sentencias PREPARE y EXECUTE.

PREPARE
: Convierte el formato de serie de caracteres de la sentencia de SQL en un formato ejecutable de la sentencia, asigna un nombre de sentencia y opcionalmente coloca información sobre la sentencia en una estructura SQLDA.

EXECUTE
: Ejecuta una sentencia de SQL previamente preparada. La sentencia se puede ejecutar repetidamente dentro de una conexión.

DESCRIBE
: Coloca información sobre una sentencia preparada en una SQLDA.

Una aplicación puede ejecutar la mayoría de las sentencias de SQL soportadas de forma dinámica.

Nota: el contenido de las sentencias de SQL dinámico siguen la misma sintaxis que las sentencias de SQL estático, con las siguientes excepciones:

- No se permiten comentarios.
- La sentencia no puede comenzar por EXEC SQL.
- La sentencia no puede finalizar con el terminador de sentencia. Una excepción a esta norma es la sentencia CREATE TRIGGER, que puede contener un signo de punto y coma (;).

Consulta relacionada:

- Apéndice A, “Sentencias de SQL soportadas” en la página 521

SQL dinámico frente a SQL estático

La pregunta sobre si utilizar SQL estático o dinámico para optimizar el rendimiento suele resultar de gran interés para los programadores. La respuesta depende de la situación.

Utilice la tabla siguiente para decidir si utilizar SQL estático o dinámico. Consideraciones como seguridad indican utilizar SQL estático, mientras que consideraciones de entorno (por ejemplo, utilizar CLI de DB2 o el CLP) indican utilizar SQL dinámico. Cuando tome su decisión, tenga en cuenta las siguientes recomendaciones sobre si se debe elegir SQL estático o dinámico en una determinada situación. En la tabla siguiente, 'Cualquiera' significa que no hay ninguna ventaja en utilizar SQL estático o dinámico.

Nota: esta tabla contiene recomendaciones generales. La aplicación específica, el uso para el que esté pensada y el entorno de trabajo dictan la opción real. Cuando tenga dudas, el mejor enfoque consiste en hacer prototipos de sus sentencias como SQL estático, luego como SQL dinámico y luego comparar las diferencias.

Tabla 8. Comparación entre SQL estático y dinámico

Consideraciones	**Probablemente la mejor opción**
Tiempo de ejecución de la sentencia de SQL:	
• Menos de 2 segundos	• Estático
• Entre 2 y 10 segundos	• Cualquiera
• Más de 10 segundos	• Dinámico
Uniformidad de datos	
• Distribución de datos uniforme	• Estático
• Ligera falta de uniformidad	• Cualquiera
• Distribución nada uniforme	• Dinámico

Tabla 8. Comparación entre SQL estático y dinámico (continuación)

Consideraciones	**Probablemente la mejor opción**
Predicados de rango (<,>,BETWEEN,LIKE)	
• Muy poco frecuentes	• Estático
• Ocasionales	• Cualquiera
• Frecuentes	• Dinámico
Ejecución repetitiva	
• Se ejecuta muchas veces (10 o más)	• Cualquiera
• Se ejecuta unas cuantas veces (menos de 10)	• Cualquiera
• Se ejecuta una vez	• Estático
Naturaleza de la consulta	
• Aleatoria	• Dinámico
• Permanente	• Cualquiera
Entorno de tiempo de ejecución (DML/DDL)	
• Proceso de transacciones (sólo DML)	• Cualquiera
• Mixto (DML y DDL - DDL afecta a los paquetes)	• Dinámico
• Mixto (DML y DDL - DDL no afecta a los paquetes)	• Cualquiera
Frecuencia de RUNSTATS	
• Muy poco frecuente	• Estático
• Regular	• Cualquiera
• Frecuente	• Dinámico

En general, una aplicación que utiliza SQL dinámico tiene un coste de partida (inicial) superior por sentencia de SQL debido a la necesidad de compilar las sentencias de SQL antes de utilizarlas. Una vez compiladas, el tiempo de ejecución de SQL dinámico comparado con SQL estático debe ser equivalente y, en algunos casos, más rápido debido a que el optimizador elige mejores planes de acceso. Cada vez que se ejecuta una sentencia dinámica, el coste de compilación inicial pierde importancia. Si varios usuarios ejecutan la misma aplicación dinámica con las mismas sentencias, el coste de compilación de la sentencia sólo se aplica a la primera aplicación que emite la sentencia.

En un entorno DML y DDL mixto, el coste de compilación correspondiente a una sentencia de SQL dinámico puede variar puesto que es posible que el sistema recompile de forma implícita la sentencia mientras se ejecuta la aplicación. En un entorno mixto, la opción entre SQL estático y dinámico debe depender también de la frecuencia con la que se invalidan paquetes. Si el DDL no invalida paquetes, es posible que SQL dinámico resulte más eficiente puesto que sólo las consultas ejecutadas se recompilan cuando se utilizan la siguiente vez. Las otras no se recompilan. Para SQL estático, el paquete entero se revincula cuando se ha invalidado.

Ahora supongamos que su aplicación particular contiene una combinación de las características anteriores y algunas de estas características sugieren que utilice SQL estático y otras que utilice SQL dinámico. En este caso, no hay ninguna decisión obvia y probablemente deba utilizar el método con el que tenga más experiencia y con el que se sienta más cómodo. Observe que las consideraciones de la tabla anterior se listan en líneas generales por orden de importancia.

Nota: tanto SQL estático como SQL dinámico vienen en dos tipos que constituyen una diferencia para el optimizador de DB2. Estos tipos son:

1. SQL estático que no contiene variables del lenguaje principal

   Esta es una situación poco probable que sólo verá para:

   - Código de *inicialización*
   - Ejemplos de formación para principiantes

   Realmente es la mejor combinación, desde la perspectiva del rendimiento, puesto que no hay proceso general de rendimiento en tiempo de ejecución y se pueden aprovechar por completo las funciones del optimizador de DB2.

2. SQL estático que contiene variables del lenguaje principal

   Es el estilo *antiguo* tradicional de las aplicaciones de DB2. Evita el proceso general en tiempo de ejecución de una sentencia PREPARE y bloqueos de catálogo adquiridos durante la compilación de sentencias. Desgraciadamente, no se puede utilizar la potencia completa del optimizador porque este no conoce la sentencia de SQL completa. Existe un problema particular con distribuciones de datos nada uniformes.

3. SQL dinámico que no contiene marcadores de parámetros

   Es el estilo típico para interfaces de consultas aleatorias (como el CLP) y es el *estilo* de SQL preferido del optimizador. Para consultas complejas, el proceso general de la sentencia PREPARE queda compensado por la mejora en el tiempo de ejecución.

4. SQL dinámico que contiene marcadores de parámetros

   Es el tipo más común de SQL para aplicaciones CLI. La ventaja clave es que la presencia de marcadores de parámetros permite amortizar el coste de la sentencia PREPARE durante ejecuciones repetidas de la sentencia, normalmente una sentencia select o insert. Esta amortización es real para todas las aplicaciones de SQL dinámico repetitivas. Desgraciadamente, al igual que SQL estático con variables del lenguaje principal, partes del optimizador de DB2 no funcionarán porque no está disponible la información completa.

La recomendación es utilizar *SQL estático con variables del lenguaje principal* o *SQL dinámico sin marcadores de parámetros* como las opciones más eficientes.

Conceptos relacionados:

- "Ejemplo de marcadores de parámetros en un programa de SQL dinámico" en la página 167

Tareas relacionadas:

- "Cómo proporcionar entrada de variables a SQL dinámico mediante marcadores de parámetros" en la página 166

Cursores en programas de SQL dinámico

Las secciones siguientes describen cómo declarar y utilizar cursores en SQL dinámico y describen brevemente los programas de ejemplo que utilizan cursores.

Declaración y utilización de cursores en programas de SQL dinámico

Procesar un cursor de forma dinámica es casi idéntico a procesarlo mediante SQL estático. Cuando se declara un cursor, se asocia con una consulta.

En SQL estático, la consulta es una sentencia SELECT en formato de texto, mientras que en SQL dinámico la consulta se asocia con un nombre de sentencia asignado en una sentencia PREPARE. Cualquier variable del lenguaje principal a la que se haga referencia se representa mediante marcadores de parámetros.

La principal diferencia entre un cursor estático y uno dinámico es que un cursor estático se prepara en el momento de la precompilación y un cursor dinámico se prepara en el tiempo de ejecución. Además, las variables del lenguaje principal a las que se hace referencia en la consulta están representadas mediante marcadores de parámetros, los cuales se sustituyen por variables del lenguaje principal de tiempo de ejecución cuando se abre el cursor.

Procedimiento:

Utilice los ejemplos que se muestran en la tabla siguiente cuando codifique cursores para un programa de SQL dinámico:

Tabla 9. Sentencia declare asociada con una sentencia SELECT dinámica

Lenguaje	Código fuente de ejemplo

C/C++	`strcpy( prep_string, "SELECT tabname FROM syscat.tables"`
	`"WHERE tabschema = ?" );`
	`EXEC SQL PREPARE s1 FROM :prep_string;`
	`EXEC SQL DECLARE c1 CURSOR FOR s1;`
	`EXEC SQL OPEN c1 USING :host_var;`

Java (JDBC)	`PreparedStatement prep_string = ("SELECT tabname FROM syscat.tables`
	`WHERE tabschema = ?" );`
	`prep_string.setCursor("c1");`
	`prep_string.setString(1, host_var);`
	`ResultSet rs = prep_string.executeQuery();`

COBOL	`MOVE "SELECT TABNAME FROM SYSCAT.TABLES WHERE TABSCHEMA = ?"`
	`TO PREP-STRING.`
	`EXEC SQL PREPARE S1 FROM :PREP-STRING END-EXEC.`
	`EXEC SQL DECLARE C1 CURSOR FOR S1 END-EXEC.`
	`EXEC SQL OPEN C1 USING :host-var END-EXEC.`

FORTRAN	`prep_string = 'SELECT tabname FROM syscat.tables WHERE tabschema = ?'`
	`EXEC SQL PREPARE s1 FROM :prep_string`
	`EXEC SQL DECLARE c1 CURSOR FOR s1`
	`EXEC SQL OPEN c1 USING :host_var`

Conceptos relacionados:

- "Ejemplo de un cursor en un programa de SQL dinámico" en la página 145
- "Cursores en REXX" en la página 381

Tareas relacionadas:

- "Selección de varias filas utilizando un cursor" en la página 118

Ejemplo de un cursor en un programa de SQL dinámico

Una sentencia de SQL dinámico se puede preparar para su ejecución con la sentencia PREPARE y se puede ejecutar con la sentencia EXECUTE o con la sentencia DECLARE CURSOR.

PREPARE con EXECUTE

El siguiente ejemplo muestra cómo se puede preparar una sentencia de SQL dinámico para su ejecución con la sentencia PREPARE y ejecutar con la sentencia EXECUTE:

- C/C++ (**dbuse.sqc/dbuse.sqC**):

  El siguiente ejemplo procede del ejemplo **dbuse**:

```
EXEC SQL BEGIN DECLARE SECTION;
  char hostVarStmt[50];
EXEC SQL END DECLARE SECTION;

strcpy(hostVarStmt, "DELETE FROM org WHERE deptnumb = 15");
EXEC SQL PREPARE Stmt FROM :hostVarStmt;
EXEC SQL EXECUTE Stmt;
```

PREPARE con DECLARE CURSOR

Los ejemplos siguientes muestran cómo se puede preparar una sentencia de SQL dinámico para su ejecución con la sentencia PREPARE y ejecutar con la sentencia DECLARE CURSOR:

- C

```
EXEC SQL BEGIN DECLARE SECTION;
  char  st[80];
  char  parm_var[19};
EXEC SQL END DECLARE SECTION;

strcpy( st, "SELECT tabname FROM syscat.tables" );
strcat( st, " WHERE tabname <> ? ORDER BY 1" );
EXEC SQL PREPARE s1 FROM :st;
EXEC SQL DECLARE c1 CURSOR FOR s1;
strcpy( parm_var, "STAFF" );
EXEC SQL OPEN c1 USING :parm_var;
```

- Java

```
PreparedStatement pstmt1 = con.prepareStatement(
  "SELECT tabname FROM syscat.tables " +
  "WHERE tabname <> ? ORDER BY 1");

// definir nombre de cursor para la sentencia update posicionada
pstmt1.setCursorName("c1");
pstmt1.setString(1, "STAFF");
ResultSet rs = pstmt1.executeQuery();
```

- COBOL **(dynamic.sqb)**

  El siguiente ejemplo procede del ejemplo **dynamic.sqb**:

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
     01 st              pic x(80).
     01 parm-var        pic x(18).
EXEC SQL END DECLARE SECTION END-EXEC.

move "SELECT TABNAME FROM SYSCAT.TABLES ORDER BY 1 WHERE TABNAME <> ?" to st.
EXEC SQL PREPARE s1 FROM :st END-EXEC.

EXEC SQL DECLARE c1 CURSOR FOR s1 END-EXEC.

move "STAFF" to parm-var.
EXEC SQL OPEN c1 USING :parm-var END-EXEC.
```

EXECUTE IMMEDIATE

También puede preparar y ejecutar una sentencia de SQL dinámico con la sentencia EXECUTE IMMEDIATE (excepto para las sentencias SELECT que devuelven más de una fila).

- C/C++ (**dbuse.sqc/dbuse.sqC**)

  El siguiente ejemplo procede de la función DynamicStmtEXECUTE_IMMEDIATE() del ejemplo **dbuse**:

```
EXEC SQL BEGIN DECLARE SECTION;
  char stmt1[50];
EXEC SQL END DECLARE SECTION;

strcpy(stmt1, "CREATE TABLE table1(col1 INTEGER)");
EXEC SQL EXECUTE IMMEDIATE :stmt1;
```

Conceptos relacionados:

- "Recuperación de mensajes de error en una aplicación" en la página 137

Ejemplos relacionados:

- "dbuse.out -- HOW TO USE A DATABASE (C)"
- "dbuse.sqc -- How to use a database (C)"
- "dbuse.out -- HOW TO USE A DATABASE (C++)"
- "dbuse.sqC -- How to use a database (C++)"

Efectos de DYNAMICRULES en SQL dinámico

La opción DYNAMICRULES de PRECOMPILE y BIND determina los valores que se aplicarán en el tiempo de ejecución para los siguientes atributos de SQL:

- El ID de autorización que se utiliza durante la comprobación de autorización.
- El calificador que se utiliza para la calificación de objetos no calificados.
- Si el paquete se puede utilizar para preparar de forma dinámica las siguientes sentencias: GRANT, REVOKE, ALTER, CREATE, DROP, COMMENT ON, RENAME, SET INTEGRITY y SET EVENT MONITOR STATE.

Además del valor de DYNAMICRULES, el entorno de tiempo de ejecución de un paquete controla el modo en que las sentencias de SQL se comportan en el tiempo de ejecución. Los dos posibles entornos de tiempo de ejecución son:

- El paquete se ejecuta como parte de un programa autónomo
- El paquete se ejecuta dentro de un contexto de rutina

La combinación del valor de DYNAMICRULES y del entorno de tiempo de ejecución determina los valores correspondientes a los atributos de SQL dinámico. Este conjunto de valores de atributos se denomina el comportamiento de las sentencias de SQL dinámico. Los cuatro comportamientos son:

Comportamiento de ejecución

DB2 utiliza el ID de autorización del usuario (el ID que inicialmente se conectó a DB2) que ejecuta el paquete como el valor que se va a utilizar para la comprobación de autorización de sentencias de SQL dinámico y para el valor inicial utilizado para la calificación implícita de referencias a objetos no calificados dentro de sentencias de SQL dinámico.

Comportamiento de vinculación

En el tiempo de ejecución, DB2 utiliza todas las normas que se aplican a SQL estático para la autorización y calificación. Es decir, se toma el ID de autorización del propietario del paquete como el valor que se va a utilizar para la comprobación de autorización de sentencias de SQL dinámico y el calificador por omisión del paquete para la calificación implícita de referencias a objetos no calificados dentro de sentencias de SQL dinámico.

Comportamiento de definición

El comportamiento de definición sólo se aplica si la sentencia de SQL dinámico está en un paquete que se ejecuta dentro de un contexto de rutina y el paquete se vinculó con DYNAMICRULES DEFINEBIND o DYNAMICRULES DEFINERUN. DB2 utiliza el ID de autorización del definidor de la rutina (no el vinculador de paquetes de la rutina) como el valor que se va a utilizar para la comprobación de autorización de sentencias de SQL dinámico y para la calificación implícita de referencias a objetos no calificados dentro de sentencias de SQL dinámico contenidas en dicha rutina.

Comportamiento de invocación

El comportamiento de invocación sólo se aplica si la sentencia de SQL dinámico está en un paquete que se ejecuta dentro de un contexto de rutina y el paquete se vinculó con DYNAMICRULES INVOKEBIND o DYNAMICRULES INVOKERUN. DB2 utiliza el ID de autorización de sentencias actualmente en vigor cuando se invoca la rutina como el valor que se va a utilizar para la comprobación de autorización de SQL dinámico y para la calificación implícita de referencias a objetos no calificados dentro de sentencias de SQL dinámico

contenidas en dicha rutina. Esto se resume en la tabla siguiente:

Entorno de invocación	ID utilizado
Cualquier SQL estático	Valor implícito o explícito del propietario (OWNER) del paquete del que procede el SQL que invoca la rutina.
Utilizado en definición de vista o activador	Definidor de la vista o activador.
SQL dinámico procedente de un paquete de comportamiento de ejecución	ID utilizado para efectuar la conexión inicial con DB2.
SQL dinámico procedente del paquete de comportamiento de definición	Definidor de la rutina que utiliza el paquete del que procede el SQL que invoca la rutina.
SQL dinámico procedente de un paquete de comportamiento de invocación	ID de autorización Current que invoca la rutina.

La tabla siguiente muestra la combinación del valor de DYNAMICRULES y el entorno de tiempo de ejecución que da lugar a cada comportamiento de SQL dinámico.

Tabla 10. Cómo DYNAMICRULES y el entorno de tiempo de ejecución determinan el comportamiento de las sentencias de SQL dinámico

Valor de DYNAMICRULES	Comportamiento de sentencias de SQL dinámico en un entorno de programa autónomo	Comportamiento de sentencias de SQL dinámico en un entorno de rutina
BIND	Comportamiento de vinculación	Comportamiento de vinculación
RUN	Comportamiento de ejecución	Comportamiento de ejecución
DEFINEBIND	Comportamiento de vinculación	Comportamiento de definición
DEFINERUN	Comportamiento de ejecución	Comportamiento de definición
INVOKEBIND	Comportamiento de vinculación	Comportamiento de invocación
INVOKERUN	Comportamiento de ejecución	Comportamiento de invocación

La tabla siguiente muestra los valores de atributos de SQL dinámico correspondientes a cada tipo de comportamiento de SQL dinámico.

Tabla 11. Definiciones de comportamientos de sentencias de SQL dinámico

Atributo de SQL dinámico	**Valor correspondiente a atributos de SQL dinámico: comportamiento de vinculación**	**Valor correspondiente a atributos de SQL dinámico: comportamiento de ejecución**	**Valor correspondiente a atributos de SQL dinámico: comportamiento de definición**	**Valor correspondiente a atributos de SQL dinámico: comportamiento de invocación**
ID de autorización	El valor implícito o explícito de la opción OWNER BIND	ID de usuario que ejecuta el paquete	Definidor de la rutina (no el propietario del paquete de la rutina)	ID de autorización de sentencias actual cuando se invoca la rutina.
Calificador por omisión para objetos no calificados	El valor implícito o explícito de la opción QUALIFIER BIND	Registro especial CURRENT SCHEMA	Definidor de la rutina (no el propietario del paquete de la rutina)	ID de autorización de sentencias actual cuando se invoca la rutina.
Puede ejecutar GRANT, REVOKE, ALTER, CREATE, DROP, COMMENT ON, RENAME, SET INTEGRITY y SET EVENT MONITOR STATE	No	Sí	No	No

Conceptos relacionados:

- "Consideraciones sobre autorización para SQL dinámico" en la página 61

El SQLDA en programas de SQL dinámico

Las secciones siguientes describen las distintas consideraciones a tener en cuenta cuando se declara el SQLDA para un programa de SQL dinámico.

Variables del lenguaje principal en el SQLDA en programas de SQL dinámico

Con SQL estático, las variables del lenguaje principal utilizadas en sentencias de SQL incorporado se conocen en el momento de compilar la aplicación. Con SQL dinámico, las sentencias de SQL incorporado y por lo tanto las variables del lenguaje principal no se conocen hasta el momento de ejecutar la

aplicación. Por lo tanto, para aplicaciones de SQL dinámico, tiene que trabajar con la lista de las variables del lenguaje principal que se utilizan en la aplicación. Puede utilizar la sentencia DESCRIBE para obtener información sobre variables del lenguaje principal para cualquier sentencia SELECT que se haya preparado (mediante PREPARE) y almacenar dicha información en el área del descriptor de SQL (SQLDA).

Nota: las aplicaciones Java no utilizan la estructura SQLDA y, por lo tanto, no utilizan las sentencias PREPARE ni DESCRIBE. En aplicaciones JDBC, puede utilizar un objeto `PreparedStatement` y el método `executeQuery()` para generar un objeto `ResultSet`, que equivale a un cursor del lenguaje principal. En aplicaciones SQLj, también puede declarar un objeto `iterador` de SQLj con un cursor `CursorByPos` o `CursorByName` para que devuelva datos de sentencias FETCH.

Cuando la sentencia DESCRIBE se ejecuta en la aplicación, el gestor de bases de datos define las variables del lenguaje principal en una SQLDA. Cuando las variables del lenguaje principal se han definido en el SQLDA, puede utilizar la sentencia FETCH para asignar valores a las variables del lenguaje principal, mediante un cursor.

Conceptos relacionados:

- "Ejemplo de un cursor en un programa de SQL dinámico" en la página 145

Consulta relacionada:

- "DESCRIBE sentencia" en el manual *Consulta de SQL, Volumen 2*
- "FETCH sentencia" en el manual *Consulta de SQL, Volumen 2*
- "PREPARE sentencia" en el manual *Consulta de SQL, Volumen 2*
- "SQLDA" en el manual *Administrative API Reference*

Declaración de la estructura SQLDA en un programa de SQL dinámico

El SQLDA contiene un número de ocurrencias de variables de entradas SQLVAR, cada una de las cuales contiene un grupo de campos que describen una columna de una fila de datos, tal como se muestra en la siguiente figura. Hay dos tipos de entradas SQLVAR: SQLVAR base y SQLVAR secundarias.

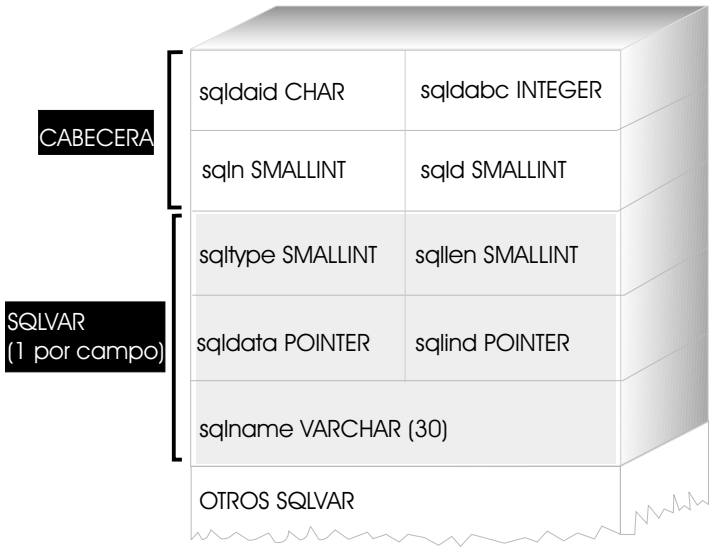


Figura 3. El Área de descriptor de SQL (SQLDA)

Procedimiento:

Puesto que el número de entradas SQLVAR necesario depende del número de columnas de la tabla de resultados, una aplicación debe ser capaz de asignar un número adecuado de elementos SQLVAR cuando se necesiten. Utilice uno de los siguientes métodos:

- Proporcione el SQLDA de mayor tamaño (es decir, la que tenga el mayor número de entradas SQLVAR) que se necesite. El número máximo de columnas que se pueden devolver en una tabla de resultados es 255. Si cualquiera de las columnas devueltas tiene un tipo LOB o un tipo diferenciado, el valor en SQLN se dobla y el número de SQLVAR necesario para albergar la información se dobla a 510. Sin embargo, puesto que la mayoría de sentencias SELECT no recuperan ni 255 columnas, la mayoría del espacio asignado no se utiliza.
- Proporcione un SQLDA de menor tamaño con menos entradas SQLVAR. En este caso, si hay más columnas en el resultado que entradas SQLVAR permitidas en el SQLDA, no se devuelve ninguna descripción. En su lugar, el gestor de bases de datos devuelve el número de elementos de lista de selección detectado en la sentencia SELECT. La aplicación asigna un SQLDA con el número de entradas SQLVAR necesario y utiliza la sentencia DESCRIBE para adquirir las descripciones de columnas.

Para ambos métodos, la duda es cuántas entradas SQLVAR iniciales se deben asignar. Cada elemento SQLVAR utiliza un máximo de 44 bytes de almacenamiento (sin contar el almacenamiento asignado para los campos

SQLDATA y SQLIND). Si hay memoria suficiente, el primer método de proporcionar un SQLDA de tamaño máximo resulta más fácil de implantar.

El segundo método de asignar un SQLDA de menor tamaño sólo se aplica a lenguajes de programación, como C y C++, que dan soporte a la asignación dinámica de memoria. Para lenguajes como COBOL y FORTRAN, que no dan soporte a la asignación dinámica de memoria, tiene que utilizar el primer método.

Tareas relacionadas:

- "Preparación de una sentencia de SQL dinámico utilizando la estructura SQLDA mínima" en la página 153
- "Asignación de un SQLDA con suficientes entradas SQLVAR para un programa de SQL dinámico" en la página 155
- "Asignación de una estructura SQLDA para un programa de SQL dinámico" en la página 158

Consulta relacionada:

- "SQLDA" en el manual *Administrative API Reference*

Preparación de una sentencia de SQL dinámico utilizando la estructura SQLDA mínima

Utilice la información de este tema como ejemplo sobre cómo asignar la estructura SQLDA mínima para una sentencia.

Restricciones:

Sólo puede asignar una estructura SQLDA de menor tamaño con lenguajes de programación, como C y C++, que den soporte a la asignación dinámica de memoria.

Procedimiento:

Supongamos que una aplicación declara una estructura SQLDA denominada `minsqlda` que no contiene ninguna entrada SQLVAR. El campo SQLN del SQLDA describe el número de entradas SQLVAR que se asignan. En este caso, SQLN se debe establecer en 0. A continuación, para preparar una sentencia desde la serie de caracteres `dstring` y para entrar su descripción en `minsqlda`, emita la siguiente sentencia de SQL (suponiendo que se utiliza sintaxis de C y que `minsqlda` se declara como un puntero a una estructura SQLDA):

```
EXEC SQL
  PREPARE STMT INTO :*minsqlda FROM :dstring;
```

Supongamos que la sentencia contenida es `dstring` es una sentencia SELECT que devuelve 20 columnas en cada fila. Después de la sentencia PREPARE (o de una sentencia DESCRIBE), el campo SQLD del SQLDA contiene el número de columnas de la tabla de resultados correspondiente a la sentencia SELECT preparada.

Las SQLVAR del SQLDA se establecen en los siguientes casos:

- SQLN >= SQLD y ninguna columna tiene el tipo LOB ni un tipo diferenciado.

  Las primeras entradas SQLVAR de SQLD se establecen y SQLDOUBLED se establece en blanco.
- SQLN >= 2*SQLD y al menos una columna es de tipo LOB o tiene un tipo diferenciado.

  2* entradas SQLVAR de SQLD se establecen y SQLDOUBLED se establece en 2.
- SQLD <= SQLN < 2*SQLD y al menos una columna tiene un tipo diferenciado, pero no hay ninguna columna LOB.

  Las primeras entradas SQLVAR de SQLD se establecen y SQLDOUBLED se establece en blanco. Si la opción de vinculación SQLWARN tiene el valor YES, se emite un aviso SQLCODE +237 (SQLSTATE 01594).

Las SQLVAR del SQLDA *no* se establecen (se necesita asignación de espacio adicional y otra sentencia DESCRIBE) en los siguientes casos:

- SQLN < SQLD y ninguna columna tiene el tipo LOB ni un tipo diferenciado.

  No se establece ninguna entrada SQLVAR y SQLDOUBLED se establece en blanco. Si la opción de vinculación SQLWARN tiene el valor YES, se emite un aviso SQLCODE +236 (SQLSTATE 01005).

  Asigne SQLVAR de SQLD para ejecutar una sentencia DESCRIBE satisfactoria.
- SQLN < SQLD y al menos una columna tiene un tipo diferenciado, pero no hay ninguna columna LOB.

  No se establece ninguna entrada SQLVAR y SQLDOUBLED se establece en blanco. Si la opción de vinculación SQLWARN tiene el valor YES, se emite un aviso SQLCODE +239 (SQLSTATE 01005).

  Asigne 2*SQLD SQLVAR para ejecutar una sentencia DESCRIBE satisfactoria, incluidos los nombres de los tipos diferenciados.
- SQLN < 2*SQLD y al menos una columna es de tipo LOB.

  No se establece ninguna entrada SQLVAR y SQLDOUBLED se establece en blanco. Se emite un aviso SQLCODE +238 (SQLSTATE 01005) (independientemente del valor de la opción de vinculación SQLWARN).

Asigne 2*SQLD SQLVAR para ejecutar una sentencia DESCRIBE satisfactoria.

La opción SQLWARN del mandato BIND sirve para controlar si DESCRIBE (o PREPARE...INTO) devolverá los siguientes avisos:

- SQLCODE +236 (SQLSTATE 01005)
- SQLCODE +237 (SQLSTATE 01594)
- SQLCODE +239 (SQLSTATE 01005).

Se recomienda que el código de la aplicación siempre tenga en cuenta que se pueden devolver estos SQLCODE. Siempre se devuelve el aviso SQLCODE +238 (SQLSTATE 01005) cuando hay columnas LOB en la lista de selección y hay un número insuficiente de SQLVAR en el SQLDA. Esta es la única manera que tiene la aplicación para saber que el número de SQLVAR se tiene que doblar debido a una columna LOB en el conjunto de resultados.

Tareas relacionadas:

- “Declaración de la estructura SQLDA en un programa de SQL dinámico” en la página 151
- “Asignación de un SQLDA con suficientes entradas SQLVAR para un programa de SQL dinámico” en la página 155
- “Asignación de una estructura SQLDA para un programa de SQL dinámico” en la página 158

Asignación de un SQLDA con suficientes entradas SQLVAR para un programa de SQL dinámico

Después de determinar el número de columnas de la tabla de resultados, asigne almacenamiento para un segundo SQLDA de tamaño completo.

Procedimiento:

Supongamos que la tabla de resultados tiene 20 columnas (ninguna de las cuales es de tipo LOB). En esta situación, debe asignar una segunda estructura SQLDA, `fulsqlda`, con un mínimo de 20 elementos SQLVAR (o 40 elementos si la tabla de resultados contiene algún LOB o tipo diferenciado). Para el resto de este ejemplo, supongamos que no hay ningún LOB ni tipo diferenciado en la tabla de resultados.

Cuando calcule los requisitos de almacenamiento para estructuras de SQLDA, incluya lo siguiente:

- Una cabecera de longitud fija, de 16 bytes de longitud, que contiene campos como SQLN y SQLD

- Una matriz de longitud variable de entradas SQLVAR, de la cual cada elemento tiene 44 bytes de longitud en plataformas de 32 bits y 56 bytes de longitud en plataformas de 64 bits.

El número de entradas SQLVAR necesarias para `fulsqlda` se especifica en el campo SQLD de `minsqlda`. Supongamos que este valor es 20. Por lo tanto, la asignación de almacenamiento necesaria para `fulsqlda` es:

```
16 + (20 * sizeof(struct sqlvar))
```

Nota: en plataformas de 64 bits, `sizeof(struct sqlvar)` y `sizeof(struct sqlvar2)` devuelven 56. En plataformas de 32 bits, `sizeof(struct sqlvar)` y `sizeof(struct sqlvar2)` devuelven 44.

Este valor representa el tamaño de la cabecera más 20 multiplicado por el tamaño de cada entrada SQLVAR, lo que da un total de 896 bytes.

Puede utilizar la macro SQLDASIZE para no tener que realizar sus propios cálculos y para evitar dependencias específicas de cada versión.

Tareas relacionadas:

- "Declaración de la estructura SQLDA en un programa de SQL dinámico" en la página 151
- "Preparación de una sentencia de SQL dinámico utilizando la estructura SQLDA mínima" en la página 153
- "Asignación de una estructura SQLDA para un programa de SQL dinámico" en la página 158

Descripción de una sentencia SELECT en un programa de SQL dinámico

Después de asignar espacio suficiente para el segundo SQLDA (en este ejemplo, denominado `fulsqlda`), debe codificar la aplicación para que describa la sentencia SELECT.

Procedimiento:

Codifique la aplicación para que lleve a cabo los pasos siguientes:

1. Almacenar el valor `20` en el campo SQLN de `fulsqlda` (en este ejemplo se supone que la tabla de resultados contiene 20 columnas y que ninguna de ellas es una columna LOB).
2. Obtener información sobre la sentencia SELECT mediante la segunda estructura SQLDA, `fulsqlda`. Hay dos métodos disponibles:
   - Utilizar otra sentencia PREPARE, especificando `fulsqlda` en lugar de `minsqlda`.
   - Utilizar la sentencia DESCRIBE especificando `fulsqlda`.

Es preferible utilizar la sentencia DESCRIBE porque se evita el coste de preparar la sentencia por segunda vez. La sentencia DESCRIBE simplemente reutiliza la información obtenida previamente durante la operación de preparación para llenar la nueva estructura SQLDA. Se puede emitir la siguiente sentencia:

```
EXEC SQL DESCRIBE STMT INTO :fulsqlda
```

Una vez ejecutada esta sentencia, cada elemento SQLVAR contiene una descripción de una columna de la tabla de resultados.

Tareas relacionadas:

- "Adquisición de almacenamiento para albergar una fila" en la página 157

Adquisición de almacenamiento para albergar una fila

Para que una aplicación pueda captar una fila de la tabla de resultados utilizando una estructura SQLDA, la aplicación debe antes asignar almacenamiento para la fila.

Procedimiento:

Codifique la aplicación de modo que haga lo siguiente:

1. Analice cada descripción de SQLVAR para determinar cuánto espacio se necesita para el valor de dicha columna.

   Tenga en cuenta que, para valores LOB, cuando se describe la sentencia SELECT, los datos que se proporcionan en la SQLVAR son SQL_TYP_*x*LOB. Este tipo de datos corresponde a una variable del lenguaje principal LOB plana, es decir, el LOB completo se almacena en memoria de una sola vez. Esto funciona para LOB pequeños (de un máximo de unos pocos MB), pero no puede utilizar este tipo de datos para LOB grandes (por ejemplo, de 1 GB). Será necesario que la aplicación cambie su definición de columna en la SQLVAR para que sea SQL_TYP_*x*LOB_LOCATOR o SQL_TYPE_*x*LOB_FILE. (Tenga en que cuenta que, si se cambia el campo SQLTYPE de la SQLVAR, también se tiene que cambiar el campo SQLLEN.) Después de cambiar la definición de columna en la SQLVAR, la aplicación puede asignar la cantidad correcta de almacenamiento correspondiente al nuevo tipo.
2. Asigne almacenamiento para el valor de dicha columna.
3. Almacene la dirección del almacenamiento asignado en el campo SQLDATA de la estructura SQLDA.

Estos pasos se deben llevar a cabo analizando la descripción de cada columna y sustituyendo el contenido de cada campo SQLDATA por la dirección de un área de almacenamiento lo suficientemente grande como para albergar cualquier valor procedente de dicha columna. El atributo de longitud se

determina a partir del campo SQLLEN de cada entrada SQLVAR correspondiente a elementos de datos que no son de tipo LOB. Para elementos con un tipo BLOB, CLOB o DBCLOB, el atributo de longitud se determina a partir del campo SQLLONGLEN de la entrada SQLVAR secundaria.

Además, si la columna especificada permite nulos, la aplicación debe sustituir el contenido del campo SQLIND por la dirección de una variable de indicador correspondiente a la columna.

Conceptos relacionados:

- "Uso de objetos grandes" en el manual *Guía de desarrollo de aplicaciones: Programación de aplicaciones de servidor*

Tareas relacionadas:

- "Proceso del cursor en un programa de SQL dinámico" en la página 158

Proceso del cursor en un programa de SQL dinámico

Después de asignar correctamente la estructura SQLDA, el cursor asociado con la sentencia SELECT se puede abrir y se pueden captar filas.

Procedimiento:

Para procesar el cursor asociado con una sentencia SELECT, primero abra el cursor y luego capte filas especificando la cláusula USING DESCRIPTOR de la sentencia FETCH. Por ejemplo, una aplicación C podría tener lo siguiente:

```
EXEC SQL OPEN pcurs
EMB_SQL_CHECK( "OPEN" ) ;
EXEC SQL FETCH pcurs USING DESCRIPTOR :*sqldaPointer
EMB_SQL_CHECK( "FETCH" ) ;
```

Para procesar una operación FETCH satisfactoriamente, podría escribir la aplicación de modo que obtuviera los datos del SQLDA y mostrara las cabeceras de columna. Por ejemplo:

```
display_col_titles( sqldaPointer ) ;
```

Una vez visualizados los datos, debería cerrar el cursor y liberar la memoria asignada de forma dinámica. Por ejemplo:

```
EXEC SQL CLOSE pcurs ;
EMB_SQL_CHECK( "CLOSE CURSOR" ) ;
```

Asignación de una estructura SQLDA para un programa de SQL dinámico

Asigne una estructura SQLDA correspondiente a la aplicación de modo que pueda utilizarla para pasar datos a la aplicación y para recibir datos de esta.

Procedimiento:

Para crear una estructura SQLDA con C, incorpore la sentencia INCLUDE SQLDA en el lenguaje principal o incluya el archivo include de SQLDA para obtener la definición de estructura. Luego, debido a que el tamaño de un SQLDA no es fijo, la aplicación debe declarar un puntero a una estructura SQLDA y asignar almacenamiento para la misma. El tamaño real de la estructura SQLDA depende del número de elementos de datos diferenciados que se pasan mediante el SQLDA.

En el lenguaje de programación C/C++, se proporciona una macro que facilita la asignación de SQLDA. Con la excepción de la plataforma HP-UX, esta macro tiene el siguiente formato:

```
#define SQLDASIZE(n) (offsetof(struct sqlda, sqlvar) \
        + (n) * sizeof(struct sqlvar))
```

En la plataforma HP-UX, la macro tiene el siguiente formato:

```
#define SQLDASIZE(n) (sizeof(struct sqlda) \
        + (n-1) * sizeof(struct sqlvar))
```

El efecto de esta macro es calcular el almacenamiento necesario para un SQLDA con `n` elementos SQLVAR.

Para crear una estructura SQLDA con COBOL, puede incorporar una sentencia INCLUDE SQLDA o utilizar la sentencia COPY. Utilice la sentencia COPY cuando desee controlar el número máximo de SQLVAR y por lo tanto la cantidad de almacenamiento que utiliza el SQLDA. Por ejemplo, para cambiar el número por omisión de SQLVAR de 1489 a 1, utilice la siguiente sentencia COPY:

```
COPY "sqlda.cbl"
  replacing --1489--
  by --1--.
```

El lenguaje FORTRAN no da soporte directamente a las estructuras de datos autodefinidas ni a la asignación dinámica. No se proporciona ningún archivo include de SQLDA para FORTRAN porque no es posible dar soporte al SQLDA como una estructura de datos en FORTRAN. El precompilador pasará por alto la sentencia INCLUDE SQLDA en un programa FORTRAN.

Sin embargo, puede crear algo parecido a una estructura SQLDA estática en un programa FORTRAN y utilizar dicha estructura siempre que se pueda utilizar un SQLDA. El archivo `sqldact.f` contiene constantes que ayudan a declarar una estructura SQLDA en FORTRAN.

Ejecute llamadas a SQLGADDR para asignar valores de puntero a elementos de SQLDA que los necesiten.

La tabla siguiente muestra la declaración y uso de una estructura SQLDA con un elemento SQLVAR.

Lenguaje **Código fuente de ejemplo**

C/C++

```
#include <sqlda.h>
struct sqlda *outda = (struct sqlda *)malloc(SQLDASIZE(1));

/* DECLARAR VARIABLES LOCALES PARA ALBERGAR DATOS REALES */
double sal;
double sal = 0;
short salind;
short salind = 0;

/* INICIALIZAR UN ELEMENTO DE SQLDA */
memcpy( outda->sqldaid,"SQLDA   ",sizeof(outda->sqldaid));
outda->sqln = outda->sqld = 1;
outda->sqlvar[0].sqltype = SQL_TYP_NFLOAT;
outda->sqlvar[0].sqllen  = sizeof( double );.
outda->sqlvar[0].sqldata = (unsigned char *)&sal;
outda->sqlvar[0].sqlind  = (short *)&salind;
```

COBOL

```
       WORKING-STORAGE SECTION.
       77 SALARY          PIC S99999V99 COMP-3.
       77 SAL-IND         PIC S9(4)     COMP-5.

          EXEC SQL INCLUDE SQLDA END-EXEC

      * O codificar un modo útil para guardar entradas SQLVAR no utilizadas.
      * COPY "sqlda.cbl" REPLACING --1489-- BY --1--.

            01 decimal-sqllen pic s9(4) comp-5.
            01 decimal-parts redefines decimal-sqllen.
               05 precision pic x.
               05 scale pic x.

      * Inicializar un elemento de SQLDA de salida
            MOVE 1 TO SQLN
            MOVE 1 TO SQLD
            MOVE SQL-TYP-NDECIMAL TO SQLTYPE(1)

      * Longitud = 7 dígitos de precisión y 2 dígitos de escala

            MOVE x"07" TO PRECISION.
            MOVE x"02" TO SCALE.
            MOVE DECIMAL-SQLLEN TO O-SQLLEN(1).
            SET SQLDATA(1) TO ADDRESS OF SALARY
            SET SQLIND(1)  TO ADDRESS OF SAL-IND
```

Lenguaje	Código fuente de ejemplo

FORTRAN

```
      include 'sqldact.f'

      integer*2  sqlvar1
      parameter ( sqlvar1 = sqlda_header_sz + 0*sqlvar_struct_sz )

C     Declarar un SQLDA de salida -- 1 Variable
      character    out_sqlda(sqlda_header_sz + 1*sqlvar_struct_sz)

      character*8  out_sqldaid     ! Cabecera
      integer*4    out_sqldabc
      integer*2    out_sqln
      integer*2    out_sqld

      integer*2    out_sqltype1    ! Primera variable
      integer*2    out_sqllen1
      integer*4    out_sqldata1
      integer*4    out_sqlind1
      integer*2    out_sqlname1
      character*30 out_sqlnamec1

      equivalence( out_sqlda(sqlda_sqldaid_ofs), out_sqldaid )
      equivalence( out_sqlda(sqlda_sqldabc_ofs), out_sqldabc )
      equivalence( out_sqlda(sqlda_sqln_ofs), out_sqln       )
      equivalence( out_sqlda(sqlda_sqld_ofs), out_sqld       )
      equivalence( out_sqlda(sqlvar1+sqlvar_type_ofs), out_sqltype1 )
      equivalence( out_sqlda(sqlvar1+sqlvar_len_ofs), out_sqllen1   )
      equivalence( out_sqlda(sqlvar1+sqlvar_data_ofs), out_sqldata1 )
      equivalence( out_sqlda(sqlvar1+sqlvar_ind_ofs), out_sqlind1   )
      equivalence( out_sqlda(sqlvar1+sqlvar_name_length_ofs),
     +             out_sqlname1                                    )
      equivalence( out_sqlda(sqlvar1+sqlvar_name_data_ofs),
     +             out_sqlnamec1                                   )

C     Declarar variables locales para albergar los datos devueltos.
      real*8       salary
      integer*2    sal_ind

C     Inicializar el SQLDA de salida (cabecera)
      out_sqldaid  = 'OUT_SQLDA'
      out_sqldabc  = sqlda_header_sz + 1*sqlvar_struct_sz
      out_sqln     = 1
      out_sqld     = 1
C     Inicializar VAR1
      out_sqltype1 = SQL_TYP_NFLOAT
      out_sqllen1  = 8
      rc = sqlgaddr( %ref(salary), %ref(out_sqldata1) )
      rc = sqlgaddr( %ref(sal_ind), %ref(out_sqlind1) )
```

En lenguajes que no dan soporte a la asignación dinámica de memoria, se debe declarar de forma explícita en el lenguaje principal un SQLDA con el

número deseado de elementos SQLVAR. Asegúrese de declarar el número suficiente de elementos SQLVAR según los requisitos de la aplicación.

Tareas relacionadas:

- "Preparación de una sentencia de SQL dinámico utilizando la estructura SQLDA mínima" en la página 153
- "Asignación de un SQLDA con suficientes entradas SQLVAR para un programa de SQL dinámico" en la página 155
- "Transferencia de datos en un programa de SQL dinámico mediante una estructura SQLDA" en la página 162

Transferencia de datos en un programa de SQL dinámico mediante una estructura SQLDA

Puede obtener una mayor flexibilidad si transfiere datos mediante un SQLDA que si utiliza listas de variables del lenguaje principal. Por ejemplo, puede utilizar un SQLDA para transferir datos que no tienen ningún equivalente en el lenguaje principal, como datos DECIMAL en lenguaje C.

Procedimiento:

Utilice la siguiente tabla como listado de referencias cruzadas que muestra cómo se relacionan los valores numéricos y los nombres simbólicos.

Tabla 12. Tipos de SQL de SQLDA de DB2. Valores numéricos y nombres simbólicos correspondientes

Tipo de columna SQL	Valor numérico SQLTYPE	Nombre simbólico SQLTYPE[1]
DATE	384 / 385	SQL_TYP_DATE / SQL_TYP_NDATE
TIME	388 / 389	SQL_TYP_TIME / SQL_TYP_NTIME
TIMESTAMP	392 / 393	SQL_TYP_STAMP / SQL_TYP_NSTAMP
n/d[2]	400 / 401	SQL_TYP_CGSTR / SQL_TYP_NCGSTR
BLOB	404 / 405	SQL_TYP_BLOB / SQL_TYP_NBLOB
CLOB	408 / 409	SQL_TYP_CLOB / SQL_TYP_NCLOB
DBCLOB	412 / 413	SQL_TYP_DBCLOB / SQL_TYP_NDBCLOB
VARCHAR	448 / 449	SQL_TYP_VARCHAR / SQL_TYP_NVARCHAR
CHAR	452 / 453	SQL_TYP_CHAR / SQL_TYP_NCHAR
LONG VARCHAR	456 / 457	SQL_TYP_LONG / SQL_TYP_NLONG
n/d[3]	460 / 461	SQL_TYP_CSTR / SQL_TYP_NCSTR
VARGRAPHIC	464 / 465	SQL_TYP_VARGRAPH / SQL_TYP_NVARGRAPH
GRAPHIC	468 / 469	SQL_TYP_GRAPHIC / SQL_TYP_NGRAPHIC
LONG VARGRAPHIC	472 / 473	SQL_TYP_LONGRAPH / SQL_TYP_NLONGRAPH

Tabla 12. Tipos de SQL de SQLDA de DB2 (continuación). Valores numéricos y nombres simbólicos correspondientes

Tipo de columna SQL	Valor numérico SQLTYPE	Nombre simbólico SQLTYPE[1]
FLOAT	480/481	SQL_TYP_FLOAT / SQL_TYP_NFLOAT
REAL[4]	480/481	SQL_TYP_FLOAT / SQL_TYP_NFLOAT
DECIMAL[5]	484/485	SQL_TYP_DECIMAL / SQL_TYP_DECIMAL
INTEGER	496/497	SQL_TYP_INTEGER / SQL_TYP_NINTEGER
SMALLINT	500/501	SQL_TYP_SMALL / SQL_TYP_NSMALL
n/d	804/805	SQL_TYP_BLOB_FILE / SQL_TYPE_NBLOB_FILE
n/d	808/809	SQL_TYP_CLOB_FILE / SQL_TYPE_NCLOB_FILE
n/d	812/813	SQL_TYP_DBCLOB_FILE / SQL_TYPE_NDBCLOB_FILE
n/d	960/961	SQL_TYP_BLOB_LOCATOR / SQL_TYP_NBLOB_LOCATOR
n/d	964/965	SQL_TYP_CLOB_LOCATOR / SQL_TYP_NCLOB_LOCATOR
n/d	968/969	SQL_TYP_DBCLOB_LOCATOR / SQL_TYP_NDBCLOB_LOCATOR

Nota: estos tipos definidos se encuentran en el archivo include sql.h del subdirectorio include del directorio sqllib. (Por ejemplo, sqllib/include/sql.h para el lenguaje de programación C.)

1. Para el lenguaje de programación COBOL, el nombre SQLTYPE no utiliza el signo de subrayado (_) sino un guión (-).
2. Es una serie gráfica terminada en nulo.
3. Es una serie de caracteres terminada en nulo.
4. La diferencia entre REAL y DOUBLE en el SQLDA es el valor de la longitud (4 u 8).
5. La precisión está en el primer byte. La escala está en el segundo byte.

Tareas relacionadas:

- "Descripción de una sentencia SELECT en un programa de SQL dinámico" en la página 156
- "Adquisición de almacenamiento para albergar una fila" en la página 157
- "Proceso del cursor en un programa de SQL dinámico" en la página 158

Proceso de sentencias interactivas de SQL en programas de SQL dinámico

Una aplicación que utiliza SQL dinámico se puede escribir de modo que procese sentencias arbitrarias de SQL. Por ejemplo, si una aplicación acepta sentencias de SQL del usuario, la aplicación debe ser capaz de ejecutar las sentencias sin conocimiento previo de las mismas.

Procedimiento:

Utilice las sentencias PREPARE y DESCRIBE con una estructura SQLDA de modo que la aplicación pueda determinar el tipo de sentencia de SQL que se ejecutan y actuar consecuentemente.

Conceptos relacionados:

- "Determinación del tipo se sentencia en programas de SQL dinámico" en la página 164

Determinación del tipo se sentencia en programas de SQL dinámico

Cuando se prepara una sentencia de SQL, la información sobre el tipo de sentencia se puede determinar examinando la estructura SQLDA. Esta información se coloca en la estructura SQLDA en el momento de la preparación de la sentencia con la cláusula INTO o emitiendo una sentencia DESCRIBE contra una sentencia preparada anteriormente.

En cualquier caso, el gestor de bases de datos coloca un valor en el campo SQLD de la estructura SQLDA que indica el número de columnas de la tabla de resultados generada por la sentencia de SQL. Si el campo SQLD contiene un cero (0), significa que la sentencia *no* es una sentencia SELECT. Puesto que la sentencia ya está preparada, se puede ejecutar inmediatamente mediante la sentencia EXECUTE.

Si la sentencia contiene marcadores de parámetros, se debe especificar la cláusula USING. La cláusula USING puede especificar una lista de variables del lenguaje principal o una estructura SQLDA.

Si el campo SQLD es mayor que cero, significa que la sentencia es una sentencia SELECT y se debe procesar tal como se describe en las siguientes secciones.

Consulta relacionada:

- "EXECUTE sentencia" en el manual *Consulta de SQL, Volumen 2*

Proceso de sentencias SELECT de lista de variables en programas de SQL dinámico

Una sentencia SELECT de *lista de variables* es una sentencia en la que el número y los tipos de columnas que se tienen que devolver no se conocen en el momento de la precompilación. En este caso, la aplicación no sabe con antelación las variables exactas del lenguaje principal que se tienen que declarar para albergar una fila de la tabla de resultados.

Procedimiento:

Para procesar una sentencia SELECT de lista de variables, codifique la aplicación de modo que haga lo siguiente:

1. Declare un SQLDA.

   Se debe utilizar una estructura SQLDA para procesar las sentencias SELECT de lista de variables.

2. PREPARE la sentencia mediante la cláusula INTO.

   Luego la aplicación determina si la estructura SQLDA declarada tiene suficientes elementos SQLVAR. Si no es así, la aplicación asigna otra estructura SQLDA con el número necesario de elementos SQLVAR y emite una sentencia DESCRIBE adicional mediante el nuevo SQLDA.

3. Asigne los elementos SQLVAR.

   Asigne almacenamiento para las variables del lenguaje principal e indicadores necesarios para cada SQLVAR. Este paso incluye la colocación de las direcciones asignadas para los datos y variables de indicador en cada elemento SQLVAR.

4. Procese la sentencia SELECT.

   Se asocia un cursor con la sentencia preparada, se abre y las filas se captan mediante la estructura SQLDA correctamente asignada.

Tareas relacionadas:

- “Declaración de la estructura SQLDA en un programa de SQL dinámico” en la página 151
- “Preparación de una sentencia de SQL dinámico utilizando la estructura SQLDA mínima” en la página 153
- “Asignación de un SQLDA con suficientes entradas SQLVAR para un programa de SQL dinámico” en la página 155
- “Descripción de una sentencia SELECT en un programa de SQL dinámico” en la página 156
- “Adquisición de almacenamiento para albergar una fila” en la página 157
- “Proceso del cursor en un programa de SQL dinámico” en la página 158

Cómo guardar peticiones de SQL procedentes de usuarios finales

Si los usuarios de la aplicación pueden emitir peticiones de SQL desde la aplicación, es posible que desee guardar dichas peticiones.

Procedimiento:

Si la aplicación permite a los usuarios guardar sentencias arbitrarias de SQL, las puede guardar en una tabla con una columna que tenga el tipo VARCHAR, LONG VARCHAR, CLOB, VARGRAPHIC, LONG VARGRAPHIC o DBCLOB. Tenga en cuenta que los tipos de datos VARGRAPHIC, LONG

VARGRAPHIC y DBCLOB sólo están disponibles en entornos de juego de caracteres de doble byte (DBCS) y de Extended UNIX Code (EUC).

Debe guardar los elementos SQL fuente, no las versiones preparadas. Esto significa que debe recuperar y luego preparar cada sentencia antes de ejecutar la versión almacenada en la tabla. Básicamente, la aplicación prepara una sentencia de SQL a partir de una serie de caracteres y ejecuta esta sentencia de forma dinámica.

Marcadores de parámetros en programas de SQL dinámico

Las secciones siguientes describen cómo utilizar marcadores de parámetros para proporcionar entrada de variables a un programa de SQL dinámico y describen brevemente los programas de ejemplo que utilizan marcadores de parámetros.

Cómo proporcionar entrada de variables a SQL dinámico mediante marcadores de parámetros

Una sentencia de SQL dinámico no puede contener variables del lenguaje principal, porque la información de las variables del lenguaje principal (tipo de datos y longitud) sólo está disponible durante la precompilación de la aplicación. En el momento de la ejecución, la información de las variables del lenguaje principal no está disponible.

En SQL dinámico, se utilizan marcadores de parámetros en lugar de variables del lenguaje principal. Los marcadores de parámetros se indican mediante un signo de interrogación (?) e indican dónde se debe sustituir una variable del lenguaje principal dentro de una sentencia de SQL.

Procedimiento:

Supongamos que la aplicación utiliza SQL dinámico y que desea que pueda realizar una operación DELETE. Una serie de caracteres que contenga un marcador de parámetros puede tener un aspecto parecido al siguiente:

```
DELETE FROM TEMPL WHERE EMPNO = ?
```

Cuando se ejecuta esta sentencia, se especifica una variable del lenguaje principal o una estructura SQLDA mediante la cláusula USING de la sentencia EXECUTE. El contenido de la variable del lenguaje principal se utiliza cuando se ejecuta la sentencia.

El marcador de parámetros adopta un tipo de datos y longitud supuestos que dependen del contexto de su uso dentro de la sentencia de SQL. Si el tipo de datos de un marcador de parámetros no resulta obvio a partir del contexto de la sentencia en la que se utiliza, utilice CAST para especificar el tipo. Dicho

marcador de parámetros se considera un a *marcador de parámetros tipificado*. Los marcadores de parámetros tipificados se tratan como una variable del lenguaje principal del tipo determinado. Por ejemplo, la sentencia `SELECT ? FROM SYSCAT.TABLES` no es válida porque DB2 no conoce el tipo de la columna de resultados. Sin embargo, la sentencia `SELECT CAST(? AS INTEGER) FROM SYSCAT.TABLES` es válida porque cast indica que el marcador de parámetros representa un INTEGER, de modo que DB2 conoce el tipo de la columna de resultados.

Si la sentencia de SQL contiene más de un marcador de parámetros, la cláusula USING de la sentencia EXECUTE debe especificar una lista de variables del lenguaje principal (una para cada marcador de parámetros) o debe identificar un SQLDA que tenga una entrada SQLVAR para cada marcador de parámetros. (Tenga en cuenta que para LOB hay dos SQLVAR por marcador de parámetros.) La lista de variables del lenguaje principal o entradas SQLVAR se comparan de acuerdo con el orden de los marcadores de parámetros en la sentencia y deben tener tipos de datos compatibles.

Nota: utilizar un marcador de parámetros con SQL dinámico es como utilizar variables del lenguaje principal con SQL estático. En cualquier caso, el optimizador no utiliza estadísticas de distribución y es posible que no elija el mejor plan de acceso.

Las normas que se aplican a marcadores de parámetros se describen con la sentencia PREPARE.

Consulta relacionada:

- "PREPARE sentencia" en el manual *Consulta de SQL, Volumen 2*

Ejemplo de marcadores de parámetros en un programa de SQL dinámico

Los siguientes ejemplos muestran cómo utilizar marcadores de parámetros en un programa de SQL dinámico:

- C/C++ (**dbuse.sqc/dbuse.sqC**)

  La función `DynamicStmtWithMarkersEXECUTEusingHostVars()` del ejemplo de lenguaje C **dbuse.sqc** muestra cómo realizar una supresión mediante un marcador de parámetro con una variable del lenguaje principal:

```
EXEC SQL BEGIN DECLARE SECTION;
  char hostVarStmt1[50];
  short hostVarDeptnumb;
EXEC SQL END DECLARE SECTION;

/* preparar la sentencia con un marcador de parámetro */
strcpy(hostVarStmt1, "DELETE FROM org WHERE deptnumb = ?");
EXEC SQL PREPARE Stmt1 FROM :hostVarStmt1;
```

```
/* ejecutar la sentencia correspondiente a hostVarDeptnumb = 15 */
hostVarDeptnumb = 15;
EXEC SQL EXECUTE Stmt1 USING :hostVarDeptnumb;
```

- JDBC (**DbUse.java**)

  La función execPreparedStatementWithParam() del ejemplo JDBC **DbUse.java** muestra cómo realizar una supresión mediante marcadores de parámetros:

```
// preparar la sentencia con marcadores de parámetros
PreparedStatement prepStmt = con.prepareStatement(
  " DELETE FROM org WHERE deptnumb <= ? AND division = ? ");

// ejecutar la sentencia
prepStmt.setInt(1, 70);
prepStmt.setString(2, "Eastern");
prepStmt.execute();

// cerrar la sentencia
prepStmt.close();
```

- COBOL (**varinp.sqb**)

  El siguiente ejemplo procede del ejemplo de COBOL **varinp.sqb** y muestra cómo utilizar un marcador de parámetro en condiciones de búsqueda y actualización:

```
   EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 pname             pic x(10).
01 dept              pic s9(4) comp-5.
01 st                pic x(127).
01 parm-var          pic x(5).
   EXEC SQL END DECLARE SECTION END-EXEC.

   move "SELECT name, dept FROM staff
-         "   WHERE job = ? FOR UPDATE OF job" to st.
   EXEC SQL PREPARE s1 FROM :st END-EXEC.

   EXEC SQL DECLARE c1 CURSOR FOR s1 END-EXEC.

   move "Mgr" to parm-var.
   EXEC SQL OPEN c1 USING :parm-var END-EXEC

   move "Clerk" to parm-var.
   move "UPDATE staff SET job = ? WHERE CURRENT OF c1" to st.
   EXEC SQL PREPARE s2 from :st END-EXEC.

* llamar a FETCH y al bucle UPDATE.
   perform Fetch-Loop thru End-Fetch-Loop
      until SQLCODE not equal 0.

   EXEC SQL CLOSE c1 END-EXEC.
```

Conceptos relacionados:

- “Recuperación de mensajes de error en una aplicación” en la página 137

Ejemplos relacionados:

- "dbuse.out -- HOW TO USE A DATABASE (C)"
- "dbuse.sqc -- How to use a database (C)"
- "dbuse.out -- HOW TO USE A DATABASE (C++)"
- "dbuse.sqC -- How to use a database (C++)"
- "DbUse.java -- How to use a database (JDBC)"
- "DbUse.out -- HOW TO USE A DATABASE (JDBC)"

Comparación entre Interfaz de nivel de llamada (CLI) de DB2 y SQL dinámico

Las secciones siguientes describen las diferencias entre CLI de DB2 y SQL dinámico, las ventajas de CLI de DB2 sobre SQL dinámico y cuándo debe utilizar CLI de DB2 o SQL dinámico.

Interfaz de nivel de llamada de DB2 (CLI) frente a SQL dinámico incorporado

Una aplicación que utiliza una interfaz de SQL incorporado necesita un precompilador para convertir las sentencias de SQL en código, que luego se puede compilar, vincular a la base de datos y ejecutar. Por el contrario, una aplicación CLI de DB2 no se tiene que precompilar ni vincular, pero utiliza un conjunto estándar de funciones para ejecutar sentencias de SQL y servicios relacionados en el tiempo de ejecución.

Esta diferencia es importante porque, tradicionalmente, los precompiladores han sido específicos para cada producto de base de datos, lo que establece una relación entre las aplicaciones y dicho producto. CLI de DB2 le permite escribir aplicaciones portables que no dependen de un determinado producto de base de datos. Esta independencia significa que las aplicaciones CLI de DB2 no se tienen que recompilar y revincular para poder acceder a distintas bases de datos de DB2, incluidas las bases de datos del sistema principal. Simplemente se conectan a la base de datos adecuada en el tiempo de ejecución.

A continuación se muestran las diferencias y similitudes entre CLI de DB2 y SQL incorporado:

- CLI de DB2 no necesita la declaración explícita de cursores. CLI de DB2 tiene un suministro de cursores que se utilizan cuando hace falta. Luego la aplicación puede utilizar el cursor generado en el modelo de captación de cursor normal para varias sentencias `SELECT` de fila y sentencias `UPDATE` y `DELETE` colocadas.
- La sentencia `OPEN` no se utiliza en CLI de DB2. En su lugar, la ejecución de `SELECT` hace que se abra un cursor de inmediato.

- A diferencia de SQL incorporado, CLI de DB2 permite el uso de marcadores de parámetros en el equivalente de la sentencia `EXECUTE IMMEDIATE` (la función `SQLExecDirect()`).
- Una sentencia `COMMIT` o `ROLLBACK` en CLI de DB2 se suele emitir mediante la función `SQLEndTran()` en lugar de ejecutarse como una sentencia de SQL, aunque esto último está permitido.
- CLI de DB2 gestiona la información relacionada con sentencias en nombre de la aplicación y proporciona un objeto abstracto para representar la información que se denomina *descriptor de contexto de sentencias*. Este descriptor de contexto elimina la necesidad de que la aplicación utilice estructuras de datos específicas del producto.
- Parecidos al descriptor de contexto de sentencias, el *descriptor de contexto de entorno* y el *descriptor de contexto de conexiones* proporcionan métodos para hacer referencia a variables globales y a información específica de la conexión. El *descriptor de contexto descriptor* describe los parámetros de una sentencia de SQL o las columnas de un conjunto de resultados.
- Las aplicaciones CLI de DB2 pueden describir parámetros de forma dinámica en una sentencia de SQL del mismo modo que las aplicaciones CLI y de SQL incorporado describen los conjuntos de resultados. Esto permite a las aplicaciones CLI procesar de forma dinámica sentencias de SQL que contienen marcadores de parámetros sin saber de forma anticipada el tipo de datos de dichos marcadores de parámetros. Cuando se prepara la sentencia de SQL, se devuelve información de descripción que detalla los tipos de datos de los parámetros.
- CLI de DB2 utiliza los valores de SQLSTATE definidos por la especificación CAE de SQL de X/Open. Aunque el formato y la mayoría de los valores son coherentes con los valores que utilizan los productos de bases de datos relacionales de IBM, hay algunas diferencias. (También hay diferencias entre los SQLSTATES de ODBC y los SQLSTATES definidos por X/Open).

A pesar de estas diferencias, hay un importante concepto común entre SQL incorporado y CLI de DB2: *CLI de DB2 puede ejecutar cualquier sentencia de SQL que se puede preparar de forma dinámica en SQL incorporado.*

Nota: CLI de DB2 también puede aceptar algunas sentencias de SQL que no se pueden preparar de forma dinámica, como sentencias compuestas de SQL.

Cada DBMS puede tener sentencias adicionales que el usuario puede preparar de forma dinámica. En este caso, CLI de DB2 pasa las sentencias directamente al DBMS. Hay una excepción: algunos DBMS pueden preparar las sentencias COMMIT y ROLLBACK de forma dinámica, pero CLI de DB2 las interceptará y las tratará como una petición `SQLEndTran()` adecuada. Sin embargo, se recomienda utilizar la función `SQLEndTran()` para especificar la sentencia COMMIT o ROLLBACK.

Consulta relacionada:

- Apéndice A, “Sentencias de SQL soportadas” en la página 521

Ventajas de CLI de DB2 sobre SQL incorporado

La interfaz CLI de DB2 tiene varias ventajas clave sobre SQL incorporado.

- Resulta ideal para un entorno cliente-servidor, en el que la base de datos de destino no se conoce cuando se crea la aplicación. Proporciona una interfaz coherente para ejecutar sentencias de SQL, independientemente del servidor de base de datos al que se conecte la aplicación.
- Aumenta la portabilidad de las aplicaciones al eliminar la dependencia de precompiladores. Las aplicaciones se distribuyen no como código fuente de SQL incorporado, que se tiene que preprocesar para cada producto de base de datos, sino como aplicaciones compiladas o bibliotecas en tiempo de ejecución.
- Las aplicaciones CLI de DB2 individuales no se tienen que vincular a cada base de datos; solo los archivos de vinculación que se suministran con CLI de DB2 se tienen que vincular una vez para todas las aplicaciones CLI de DB2. Esto puede reducir significativamente la cantidad de gestión necesaria para la aplicación cuando ya está en nivel de uso general.
- Las aplicaciones CLI de DB2 se pueden conectar a varias bases de datos, incluidas varias conexiones a la misma base de datos, todo desde la misma aplicación. Cada conexión tiene su propio ámbito de confirmación. Esto resulta más sencillo si se utiliza CLI que si se utiliza SQL incorporado, donde la aplicación debe utilizar varias hebras para conseguir el mismo resultado.
- CLI de DB2 elimina la necesidad de áreas de datos controladas por la aplicación y a menudo complejas, como SQLDA y SQLCA, que suelen estar asociadas con aplicaciones de SQL incorporado. En su lugar, CLI de DB2 asigna y controla las estructuras de datos necesarias y proporciona un *descriptor de contexto* para que la aplicación haga referencia a las mismas.
- CLI de DB2 permite el desarrollo de aplicaciones de varias hebras seguras en las que cada hebra puede tener su propia conexión y un ámbito de confirmación independiente del resto. CLI de DB2 lo consigue eliminando las áreas de datos descritas anteriormente y asociando todas estas estructuras de datos a las que puede acceder la aplicación con un descriptor de contexto específico. A diferencia de SQL incorporado, una aplicación CLI de varias hebras no tiene que llamar a ninguna de las API de DB2 de gestión de contexto; el controlador de CLI de DB2 lo maneja automáticamente.
- CLI de DB2 proporciona una función mejorada de captación y entrada de parámetros que permite especificar matrices de datos como entrada, recuperar varias filas de un conjunto de resultados directamente en una matriz y ejecutar sentencias que generan varios conjuntos de resultados.

- CLI de DB2 proporciona una interfaz coherente frente a la información de catálogo de consulta (Tablas, Columnas, Claves foráneas, Claves principales, etc.) contenida en distintas tablas de catálogo de DBMS. Los conjuntos de resultados devueltos son coherentes entre los DBMS. Esto protege la aplicación frente a cambios de catálogo entre releases de servidores de datos, así como frente a diferencias de catálogo entre distintos servidores de bases de datos; por lo tanto evita que las aplicaciones tengan que escribir consultas de catálogo específicas de cada versión y específicas de cada servidor.
- CLI de DB2 también proporciona conversión de datos ampliados, por lo que se necesita menos código de aplicación cuando se convierte información entre distintos tipos de datos SQL y C.
- CLI de DB2 incorpora funciones CLI tanto de ODBC como de X/Open (ambas son especificaciones aceptadas de la industria). CLI de DB2 también cumple con el estándar de CLI de ISO. El conocimiento que los programadores de aplicaciones adquieran en estas especificaciones se puede aplicar directamente al desarrollo de CLI de DB2 y viceversa. Esta interfaz resulta intuitiva para los programadores que están familiarizados con bibliotecas de funciones pero tienen pocos conocimientos sobre métodos específicos del producto para incorporar sentencias de SQL en un lenguaje principal.
- CLI de DB2 proporciona la posibilidad de recuperar varias filas y conjuntos de resultados generados a partir de un procedimiento almacenado que reside en un servidor DB2 Universal Database (o DB2 Universal Database para OS/390 y z/OS versión 5 o posterior). Sin embargo, tenga en cuenta que esta función existe para clientes de la Versión 5 de DB2 Universal Database que utilizan SQL incorporado si el procedimiento almacenado reside en un servidor al que se puede acceder desde un servidor DataJoiner Versión 2.
- CLI de DB2 ofrece soporte más amplio para cursores desplazables. Con cursores desplazables puede desplazarse por un cursor del siguiente modo:
  - Hacia adelante, una o más filas
  - Hacia atrás, una o más filas
  - Desde la primera fila una o más filas.
  - Desde la última fila una o más filas.

  Los cursores desplazables se pueden utilizar junto con una salida de matriz. Puede declarar un cursor actualizable como desplazable y luego desplazarse hacia adelante o hacia atrás a través del conjunto de resultados una o más filas. También puede captar filas especificando un desplazamiento desde:
  - La fila actual
  - El principio o fin del conjunto de resultados
  - Una fila específica que ha establecido anteriormente con una marca.

Cuándo utilizar CLI de DB2 o SQL incorporado

La interfaz elegida depende de la aplicación.

CLI de DB2 resulta ideal para aplicaciones de interfaz gráfica de usuario (GUI) basada en consulta que necesitan portabilidad. Las ventajas explicadas anteriormente pueden hacer parecer que utilizar CLI de DB2 sea la opción obvia para cualquier aplicación. Sin embargo, hay un factor que se debe tener en cuenta: la comparación entre SQL estático y dinámico. Es mucho más fácil utilizar SQL estático en aplicaciones incorporadas.

SQL estático tiene varias ventajas:

- Rendimiento

  SQL dinámico se prepara en el tiempo de ejecución, SQL estático se prepara en el momento de la precompilación. Además de necesitar más proceso, el paso de preparación puede originar tráfico de red adicional en el tiempo de ejecución. El tráfico de red adicional se puede evitar si la aplicación CLI de DB2 utiliza la preparación diferida (que es el comportamiento por omisión).

  Es importante indicar que SQL estático no siempre tiene un rendimiento mejor que SQL dinámico. SQL dinámico se prepara en el tiempo de ejecución y utiliza las estadísticas de bases de datos disponibles en ese momento, mientras que SQL estático utiliza las estadísticas de bases de datos disponibles en el momento de realizar la operación BIND. SQL dinámico utiliza los cambios realizados en la base de datos, como nuevos índices, para elegir el plan de acceso óptimo, lo que potencialmente ofrece un mejor rendimiento que el mismo SQL ejecutado como SQL estático. Además, se puede evitar la precompilación de sentencias de SQL dinámico si se colocan en antememoria.
- Encapsulación y seguridad

  En SQL estático, las autorizaciones para acceder a objetos (como una tabla o una vista) están asociadas a un paquete y se validan en el momento de vincular el paquete. Esto significa que los administradores de bases de datos sólo tienen que otorgar autorización de ejecución sobre un determinado paquete a un conjunto de usuarios (encapsulando así sus privilegios en el paquete) sin tener que otorgarles acceso explícito a cada objeto de la base de datos. En SQL dinámico, las autorizaciones se validan en el tiempo de ejecución sentencia a sentencia; por lo tanto, se tiene que otorgar a los usuarios acceso explícito a cada objeto de la base de datos. Esto permite que estos usuarios accedan a partes del objeto a las que no tienen necesidad de acceder.
- SQL incorporado recibe soporte en otros lenguajes, además de C o C++.
- Para selecciones fijas de consulta, SQL incorporado resulta más sencillo.

Si una aplicación necesita las ventajas de ambas interfaces, se puede utilizar SQL estático dentro de una aplicación CLI de DB2 creando un procedimiento almacenado que contenga el SQL estático. Se llama al procedimiento almacenado desde dentro de una aplicación CLI de DB2 y se ejecuta en el servidor. Una vez creado el procedimiento almacenado, cualquier aplicación CLI de DB2 o ODBC puede llamarlo.

También se puede escribir una aplicación mixta que utilice tanto CLI de DB2 como SQL incorporado, aprovechando sus respectivas ventajas. En este caso, se utiliza CLI de DB2 para proporcionar la aplicación base, con módulos clave escritos utilizando SQL estático por razones de rendimiento o seguridad. Esto complica el diseño de la aplicación y sólo se debe utilizar si los procedimientos almacenados no cumplen con los requisitos de la aplicación.

Por último, la decisión sobre cuándo utilizar cada interfaz se basará en preferencias individuales y en la experiencia más que en cualquier otro factor.

Conceptos relacionados:

- "Recurso de rastreo de CLI/ODBC/JDBC" en la página 313

Tareas relacionadas:

- "Preparing and Executing SQL Statements in CLI Applications" en el manual *CLI Guide and Reference, Volume 1*
- "Issuing SQL Statements in CLI Applications" en el manual *CLI Guide and Reference, Volume 1*
- "Creating Static SQL with CLI/ODBC/JDBC Static Profiling" en el manual *CLI Guide and Reference, Volume 1*

Capítulo 6. Programación en C y C++

Consideraciones sobre programación en C/C++

En los temas siguientes se tratan las consideraciones especiales sobre la programación en lenguajes principales. Se incluye información sobre las restricciones de lenguaje, los archivos include específicos del lenguaje principal, la incorporación de sentencias de SQL y los tipos de datos soportados para las variables del lenguaje principal.

Consulta relacionada:

- "Programas de ejemplo C/C++" en el manual *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

Secuencias tri-grafo para C y C++

Algunos caracteres del juego de caracteres de C o C++ no están disponibles en todos los teclados. Dichos caracteres se pueden entrar en un programa fuente C o C++ utilizando una secuencia de tres caracteres denominada *tri-grafo*. Los tri-grafos no se reconocen en las sentencias de SQL. El precompilador reconoce los tri-grafos siguientes dentro de las declaraciones de variables del lenguaje principal:

Tri-grafo	Definición
??(	Corchete izquierdo '['
??)	Corchete derecho ']'
??<	Llave izquierda '{'
??>	Llave derecha '}'

Los tri-grafos restantes que se listan a continuación se pueden producir en cualquier lugar de un programa fuente C o C++:

Tri-grafo	Definición
??=	Almohadilla '#'
??/	Barra inclinada invertida '\'
??'	Signo de intercalación '^'
??!	Barra vertical '|'
??–	Tilde '~'

Archivos de entrada y de salida para C y C++

Por omisión, el archivo de entrada puede tener las extensiones siguientes:

.sqc Para los archivos C en todas las plataformas soportadas

.sqC Para archivos C++ en plataformas UNIX

.sqx Para archivos C++ en sistemas operativos Windows

Por omisión, los archivos de salida del precompilador correspondiente tienen las siguientes extensiones:

.c Para los archivos C en todas las plataformas soportadas

.C Para los archivos C++ en plataformas UNIX

.cxx Para archivos C++ en sistemas operativos Windows

Puede utilizar la opción de precompilación OUTPUT para alterar temporalmente el nombre y la vía de acceso del archivo fuente de salida modificado. Si utiliza las opciones de precompilación TARGET C o TARGET CPLUSPLUS, no es necesario que el archivo de entrada tenga ninguna extensión concreta.

Archivos include

Las siguientes secciones describen los archivos include correspondientes a C y C++.

Archivos include para C y C++

Los archivos include específicos del lenguaje principal (archivos de cabecera) para C y C++ tienen la extensión de archivo `.h`. A continuación se describen los archivos include destinados a su uso en las aplicaciones del usuario.

SQL (sql.h)
Este archivo incluye prototipos específicos del lenguaje para el vinculador, el precompilador y las API de recuperación de mensajes de error. También define constantes del sistema.

SQLADEF (sqladef.h)
Este archivo contiene prototipos de funciones utilizados por aplicaciones C y C++ precompiladas.

SQLAPREP (sqlaprep.h)
Este archivo contiene definiciones necesarias para escribir su propio precompilador.

SQLCA (sqlca.h)
Este archivo define la estructura del Área de comunicaciones de SQL (SQLCA). El SQLCA contiene variables que utiliza el gestor de bases de datos para proporcionar a una aplicación información de error sobre la ejecución de sentencias de SQL y llamadas a API.

SQLCLI (sqlcli.h)
Este archivo contiene los prototipos y constantes de funciones necesarios para escribir una aplicación de Interfaz de nivel de llamada (CLI de DB2). Las funciones de este archivo son comunes para la Interfaz de nivel de llamada de X/Open y el Nivel esencial de ODBC.

SQLCLI1 (sqlcli1.h)
Este archivo contiene los prototipos y las constantes de funciones necesarios para escribir una Interfaz de nivel de llamada (CLI de DB2) que hace uso de las características más avanzadas de la CLI de DB2. Muchas de las funciones de este archivo son comunes para la Interfaz de nivel de llamada de X/Open y el Nivel 1 de ODBC. Además, este archivo también incluye funciones sólo de X/Open y funciones específicas de DB2.

Este archivo incluye tanto `sqlcli.h` como `sqlext.h` (que contiene definiciones de API de Nivel 2 de ODBC).

SQLCODES (sqlcodes.h)
Este archivo define constantes para el campo SQLCODE de la estructura SQLCA.

SQLDA (sqlda.h)
Este archivo define la estructura del Área de descriptor de SQL (SQLDA). El SQLDA se utiliza para pasar datos entre una aplicación y el gestor de bases de datos.

SQLEAU (sqleau.h)
Este archivo contiene definiciones de constantes y de estructuras necesarias para las API de auditoría de seguridad de DB2. Si utiliza estas API, tiene que incluir este archivo en el programa. Este archivo también contiene definiciones de constantes y de valores de palabras clave para los campos del registro de seguimiento de auditoría. Programas externos o de extracción de seguimiento de auditoría de proveedores pueden utilizar estas definiciones.

SQLENV (sqlenv.h)
Este archivo define llamadas específicas del lenguaje para las API del entorno de bases de datos y las estructuras, constantes y códigos de retorno correspondientes a dichas interfaces.

SQLEXT (sqlext.h)
Este archivo contiene los prototipos y constantes de funciones de aquellas API de Nivel 1 y Nivel 2 de ODBC que no forman parte de la especificación de la Interfaz de nivel de llamada de X/Open y, por lo tanto, se utiliza con permiso de Microsoft Corporation.

SQLE819A (sqle819a.h)
Si la página de códigos de la base de datos es 819 (ISO Latin-1), esta secuencia clasifica series de caracteres que no son FOR BIT DATA

según la clasificación binaria CCSID 500 (EBCDIC internacional) del sistema principal. La API CREATE DATABASE utiliza este archivo.

SQLE819B (sqle819b.h)
Si la página de códigos de la base de datos es 819 (ISO Latin-1), esta secuencia clasifica series de caracteres que no son FOR BIT DATA según la clasificación binaria CCSID 037 (EBCDIC inglés de EE.UU.) del sistema principal. La API CREATE DATABASE utiliza este archivo.

SQLE850A (sqle850a.h)
Si la página de códigos de la base de datos es 850 (ASCII Latin-1), esta secuencia clasifica series de caracteres que no son FOR BIT DATA según la clasificación binaria CCSID 500 (EBCDIC internacional) del sistema principal. La API CREATE DATABASE utiliza este archivo.

SQLE850B (sqle850b.h)
Si la página de códigos de la base de datos es 850 (ASCII Latin-1), esta secuencia clasifica series de caracteres que no son FOR BIT DATA según la clasificación binaria CCSID 037 (EBCDIC inglés de EE.UU.) del sistema principal. La API CREATE DATABASE utiliza este archivo.

SQLE932A (sqle932a.h)
Si la página de códigos de la base de datos es 932 (ASCII japonés), esta secuencia clasifica series de caracteres que no son FOR BIT DATA según la clasificación binaria CCSID 5035 (EBCDIC japonés) del sistema principal. La API CREATE DATABASE utiliza este archivo.

SQLE932B (sqle932b.h)
Si la página de códigos de la base de datos es 932 (ASCII japonés), esta secuencia clasifica series de caracteres que no son FOR BIT DATA según la clasificación binaria CCSID 5026 (EBCDIC japonés) del sistema principal. La API CREATE DATABASE utiliza este archivo.

SQLJACB (sqljacb.h)
Este archivo define constantes, estructuras y bloques de control correspondientes a la interfaz de DB2 Connect.

SQLMON (sqlmon.h)
Este archivo define llamadas específicas del lenguaje para las API del supervisor del sistema de bases de datos y las estructuras, constantes y códigos de retorno correspondientes a dichas interfaces.

SQLSTATE (sqlstate.h)
Este archivo define constantes correspondientes al campo SQLSTATE de la estructura SQLCA.

SQLSYSTM (sqlsystm.h)
Este archivo contiene definiciones específicas de la plataforma que utilizan las API y las estructuras de datos del gestor de bases de datos.

SQLUDF (sqludf.h)
Este archivo define constantes y estructuras de interfaces para escribir funciones definidas por el usuario (UDF).

SQLUTIL (sqlutil.h)
Este archivo define las llamadas específicas del lenguaje correspondientes a las API de programas de utilidad y las estructuras, constantes y códigos necesarios para dichas interfaces.

SQLUV (sqluv.h)
Este archivo define estructuras, constantes y prototipos para la API asíncrona de Registro de lecturas y para las API utilizadas por los proveedores de carga y descarga de tablas.

SQLUVEND (sqluvend.h)
Este archivo define estructuras, constantes y prototipos correspondientes a las API que utilizarán los proveedores de gestión de almacenamiento.

SQLXA (sqlxa.h)
Este archivo contiene prototipos y constantes de funciones utilizados por las aplicaciones que usan la Interfaz XA de X/Open.

Conceptos relacionados:

• "Archivos include en C y C++" en la página 180

Archivos include en C y C++

Existen dos métodos para incluir archivos: la sentencia EXEC SQL INCLUDE y la macro #include. El precompilador pasará por alto #include y sólo procesará los archivos incluidos mediante la sentencia EXEC SQL INCLUDE.

Para localizar los archivos incluidos mediante `EXEC SQL INCLUDE`, el precompilador C de DB2 busca en primer lugar en el directorio actual y, a continuación, en los directorios especificados por la variable de entorno DB2INCLUDE. Considere los ejemplos siguientes:

- `EXEC SQL INCLUDE payroll;`

  Si el archivo especificado en la sentencia INCLUDE no está encerrado entre comillas, tal como en el ejemplo anterior, el precompilador C busca `payroll.sqc`, y luego `payroll.h`, en cada uno de los directorios que mira. En los sistemas operativos UNIX, el precompilador C++ busca `payroll.sqC`, luego `payroll.sqx`, luego `payroll.hpp` y luego `payroll.h` en cada uno de los directorios que mira. En los sistemas operativos Windows de 32 bits, el

precompilador C++ busca `payroll.sqx`, luego `payroll.hpp` y luego `payroll.h` en cada uno de los directorios que mira.

- `EXEC SQL INCLUDE 'pay/payroll.h';`

  Si el nombre de archivo está encerrado entre comillas, tal como en el caso anterior, no se añade ninguna extensión al nombre.

  Si el nombre del archivo entrecomillado no contiene una vía de acceso absoluta, se utiliza el contenido de DB2INCLUDE para buscar el archivo, añadiéndole como prefijo la vía de acceso especificada en el nombre del archivo de INCLUDE. Por ejemplo, en los sistemas basados en UNIX, si DB2INCLUDE se establece en '`/disk2:myfiles/c`', el precompilador C/C++ busca '`./pay/payroll.h`', luego '`/disk2/pay/payroll.h`' y finalmente '`./myfiles/c/pay/payroll.h`'. En los mensajes del precompilador se muestra la vía de acceso en que se encuentra realmente el archivo. En los sistemas operativos basados en Windows, sustituya las barras inclinadas invertidas (\) del ejemplo anterior por barras inclinadas.

Nota: el procesador de línea de mandatos coloca en antememoria el valor de DB2INCLUDE. Para cambiar el valor de DB2INCLUDE después de haber emitido mandatos del CLP, entre el mandato TERMINATE, luego vuelva a conectar con la base de datos y realice una precompilación del modo habitual.

Como ayuda para relacionar los errores del compilador con la fuente original, el precompilador genera macros ANSI #line en el archivo de salida. Esto permite que el compilador informe de los errores utilizando el nombre de archivo y el número de línea del archivo fuente o fuente incluido, en lugar de la salida del precompilador.

Sin embargo, si se especifica la opción PREPROCESSOR, todas las macros #line generadas por el precompilador hacen referencia al archivo preprocesado por el preprocesador C externo.

Algunos depuradores y otras herramientas que relacionan código fuente con código objeto no siempre funcionan bien con la macro #line. Si la herramienta que desea utilizar se comporta de forma inesperada, utilice la opción NOLINEMACRO (usada con DB2 PREP) cuando realice la precompilación. Esta opción evita que se generen macros #line.

Conceptos relacionados:

- "Expansión de macros en C" en la página 200

Consulta relacionada:

- "PREPARE sentencia" en el manual *Consulta de SQL, Volumen 2*
- "Archivos include para C y C++" en la página 177

Sentencias de SQL incorporado en C y C++

Las sentencias de SQL incorporado constan de los tres elementos siguientes:

Elemento	Sintaxis correcta
Inicializador de la sentencia	EXEC SQL
Serie de la sentencia	Cualquier sentencia de SQL válida
Terminador de la sentencia	punto y coma (;)

Por ejemplo:

```
EXEC SQL SELECT col INTO :hostvar FROM table;
```

Se aplican las normas siguientes a las sentencias de SQL incorporado:

- Puede empezar la serie de la sentencia de SQL en la misma línea que el par de palabras clave o en una línea separada. La serie de la sentencia puede tener una longitud de varias líneas. No divida el par de palabras clave EXEC SQL en distintas líneas.
- Debe utilizar el terminador de sentencias de SQL. De no utilizarlo, el precompilador seguirá hasta el siguiente terminador de la aplicación. Esto puede producir errores indeterminados.

  Se pueden colocar comentarios de C/C++ delante del inicializador de la sentencia o detrás del terminador de la sentencia.
- Se pueden poner varias sentencias de SQL y de C/C++ en la misma línea. Por ejemplo:

  ```
  EXEC SQL OPEN c1; if (SQLCODE >= 0) EXEC SQL FETCH c1 INTO :hv;
  ```
- El precompilador SQL deja tal cual los retornos de carro, saltos de línea y tabulaciones que aparecen en una serie entrecomillada.
- Están permitidos los comentarios de SQL en cualquier línea que forme parte de una sentencia de SQL incorporado. Estos comentarios no están permitidos en las sentencias que se ejecutan de forma dinámica. El formato de un comentario de SQL consiste en un guión doble (--) seguido de una serie compuesta por cero o más caracteres y terminado por un fin de línea. No coloque comentarios de SQL después del terminador de sentencia de SQL. Los comentarios después del terminador generan errores de compilación porque parece que formen parte del lenguaje C/C++.

  Puede utilizar comentarios en la serie de una sentencia estática siempre que se permitan caracteres en blanco. Utilice los delimitadores de comentarios de C/C++ /* */ o el símbolo de comentario de SQL (--). Los comentarios de C++ del estilo // no están permitidos en las sentencias de SQL estático, pero se pueden utilizar en cualquier otro lugar del programa. El precompilador elimina los comentarios antes de procesar la sentencia de

SQL. **No puede** utilizar los delimitadores de comentario de C y C++ /* */ ni // en una sentencia de SQL dinámico. Sin embargo, los puede utilizar en cualquier otro lugar del programa.

- Puede continuar los literales de la serie de SQL y los identificadores delimitados con divisiones de línea en las aplicaciones C y C++. Para ello, utilice una barra inclinada invertida (\) al final de la línea en que desea que se produzca la división. Por ejemplo:

```
EXEC SQL SELECT "NA\
ME" INTO :n FROM staff WHERE name='Sa\
nders';
```

  Los caracteres de línea nueva (como, por ejemplo, el retorno de carro y el salto de línea) no se incluyen en la serie ni en el identificador delimitado.
- La sustitución de caracteres de espacio en blanco, como caracteres de fin de línea y de tabulación, se produce del siguiente modo:
  - Cuando aparecen fuera de las comillas (pero dentro de sentencias de SQL), los caracteres de fin de línea y tabuladores se sustituyen por un solo espacio.
  - Si aparecen dentro de las comillas, los caracteres de fin de línea desaparecen, siempre que se continúe correctamente la serie para un programa C. Los tabuladores no se modifican.

  Observe que los caracteres reales utilizados como fin de línea y tabulador varían en función de la plataforma. Por ejemplo, los sistemas basados en UNIX utilizan un salto de línea.

Consulta relacionada:

- Apéndice A, “Sentencias de SQL soportadas” en la página 521

Variables del lenguaje principal en C y C++

Las secciones siguientes describen cómo declarar y utilizar variables del lenguaje principal en programas C y C++.

Variables del lenguaje principal en C y C++

Las variables del lenguaje principal son variables de los lenguajes C o C++ a las que se hace referencia en las sentencias de SQL. Permiten que una aplicación pase datos de entrada al gestor de bases de datos y reciba datos de salida de éste. Una vez que se ha precompilado la aplicación, el compilador utiliza las variables del lenguaje principal como cualquier otra variable de C/C++. Cuando denomine, declare y utilice variables del lenguaje principal, siga las normas descritas en los apartados siguientes.

En aplicaciones que construyen del SQLDA de forma manual, no se pueden utilizar variables tipo long cuando sqlvar::sqltype==SQL_TYP_INTEGER. En su lugar se deben utilizar tipos sqlint32. Este problema es idéntico a utilizar variables long en declaraciones de variables del lenguaje principal, excepto en que con una SQLDA construida de forma manual el precompilador no revelará este error y se producirán errores en el tiempo de ejecución.

Las difusiones de long y unsigned long que se utilizan para acceder a información sqlvar::sqldata se deben cambiar por sqlint32 y sqluint32. Los miembros val correspondientes a las estructuras sqloptions y sqla_option se declaran como sqluintptr. Por lo tanto, la asignación de miembros de puntero en miembros sqla_option::val o sqloptions::val deben utilizar difusiones sqluintptr en lugar de difusiones unsigned long. Este cambio no ocasionará problemas en el tiempo de ejecución en plataformas UNIX de 64 bits, pero se debe realizar como preparación para aplicaciones Windows NT de 64 bits, en las que el tipo long sólo es de 32 bits.

Conceptos relacionados:

- “Nombres de variables del lenguaje principal en C y C++” en la página 185
- “Declaraciones de variables del lenguaje principal en C y C++” en la página 186
- “Sintaxis de las variables del lenguaje principal de tipo carácter fijas y terminadas en nulo en C y C++” en la página 189
- “Variables de indicador en C y C++” en la página 191
- “Variables gráficas del lenguaje principal en C y C++” en la página 191
- “Inicialización de variables del lenguaje principal en C y C++” en la página 200
- “Soporte de estructuras del lenguaje principal en C y C++” en la página 201
- “Sección declare de SQL con variables del lenguaje principal para C y C++” en la página 216

Consulta relacionada:

- “Sintaxis de las variables numéricas del lenguaje principal en C y C++” en la página 187
- “Sintaxis de las variables del lenguaje principal de tipo carácter de longitud variable en C o C++” en la página 190
- “Sintaxis de las variables del lenguaje principal de objeto grande (LOB) en C o C++” en la página 195
- “Sintaxis de las variables del lenguaje principal de localizador de objeto grande (LOB) en C o C++” en la página 198
- “Sintaxis de declaraciones de variables del lenguaje principal de referencia de archivos en C o C++” en la página 199

Nombres de variables del lenguaje principal en C y C++

El precompilador SQL identifica las variables del lenguaje principal por el nombre declarado para éstas. Se aplican las normas siguientes:

- Especifique nombres de variables con una longitud máxima de 255 caracteres.
- Empiece los nombres de variables con prefijos que no sean SQL, sql, DB2 ni db2, los cuales están reservados para uso del sistema. Por ejemplo:

```
EXEC SQL BEGIN DECLARE SECTION;
  char  varsql;     /* permitido */
  char  sqlvar;     /* no permitido */
  char  SQL_VAR;    /* no permitido */
EXEC SQL END DECLARE SECTION;
```

- El precompilador considera los nombres de variables del lenguaje principal como globales respecto a un módulo. Sin embargo, esto no significa que las variables del lenguaje principal se tengan que definir como variables globales; es perfectamente aceptable que se declaren variables del lenguaje principal como variables locales dentro de funciones. Por ejemplo, el código siguiente funcionará correctamente:

```
void f1(int i)
{
EXEC SQL BEGIN DECLARE SECTION;
  short host_var_1;
EXEC SQL END DECLARE SECTION;
EXEC SQL SELECT COL1 INTO :host_var_1 from TBL1;
}
void f2(int i)
{
EXEC SQL BEGIN DECLARE SECTION;
  short host_var_2;
EXEC SQL END DECLARE SECTION;
EXEC SQL INSERT INTO TBL1 VALUES (:host_var_2);
}
```

También es posible tener varias variables del lenguaje principal locales con el mismo nombre, siempre que sean del mismo tipo y tamaño. Para ello, declare la primera aparición de la variable del lenguaje principal, ante el precompilador, entre sentencias BEGIN DECLARE SECTION y END DECLARE SECTION, y deje las declaraciones posteriores de la variable fuente de las secciones de declaración. El código siguiente muestra un ejemplo de este caso:

```
void f3(int i)
{
EXEC SQL BEGIN DECLARE SECTION;
  char host_var_3[25];
EXEC SQL END DECLARE SECTION;
EXEC SQL SELECT COL2 INTO :host_var_3 FROM TBL2;
}
void f4(int i)
```

```
{
char host_var_3[25];
EXEC SQL INSERT INTO TBL2 VALUES (:host_var_3);
}
```

Puesto que `f3` y `f4` están en el mismo módulo y `host_var_3` tiene el mismo tipo y longitud en ambas funciones, basta una sola declaración ante el precompilador para utilizarla en ambos lugares.

Conceptos relacionados:

- "Declaraciones de variables del lenguaje principal en C y C++" en la página 186

Declaraciones de variables del lenguaje principal en C y C++

Se debe utilizar una sección de declaración de SQL para identificar las declaraciones de variables del lenguaje principal. Esto alerta al precompilador acerca de las variables del lenguaje principal a las que se puede hacer referencia en sentencias de SQL posteriores.

El precompilador C/C++ sólo reconoce un subconjunto de declaraciones de C o C++ válidas como declaraciones de variables del lenguaje principal válidas. Estas declaraciones definen variables numéricas o de tipo carácter. No se admiten las definiciones de tipo para los tipos de variable del lenguaje principal. Las variables del lenguaje principal se pueden agrupar en una sola estructura de lenguaje principal. Puede declarar miembros de datos de clase C++ como variables del lenguaje principal.

Una variable numérica del lenguaje principal se puede utilizar como variable de entrada o de salida para cualquier valor numérico de entrada o salida de SQL. Una variable del lenguaje principal de tipo carácter se puede utilizar como variable de entrada o de salida para cualquier valor de tipo carácter, de fecha, de hora o de indicación horaria, de entrada o salida de SQL. La aplicación se tiene que asegurar de que las variables de salida sean lo suficientemente largas como para contener los valores que reciben.

Conceptos relacionados:

- "Sintaxis de las variables del lenguaje principal de tipo carácter fijas y terminadas en nulo en C y C++" en la página 189
- "Variables gráficas del lenguaje principal en C y C++" en la página 191
- "Soporte de estructuras del lenguaje principal en C y C++" en la página 201
- "Miembros de datos de clase utilizados como variables del lenguaje principal en C y C++" en la página 208

Tareas relacionadas:

- “Declaración de variables del lenguaje principal con el Generador de declaraciones db2dclgn” en la página 38
- “Declaración de variables del lenguaje principal de tipo estructurado” en el manual *Guía de desarrollo de aplicaciones: Programación de aplicaciones de servidor*

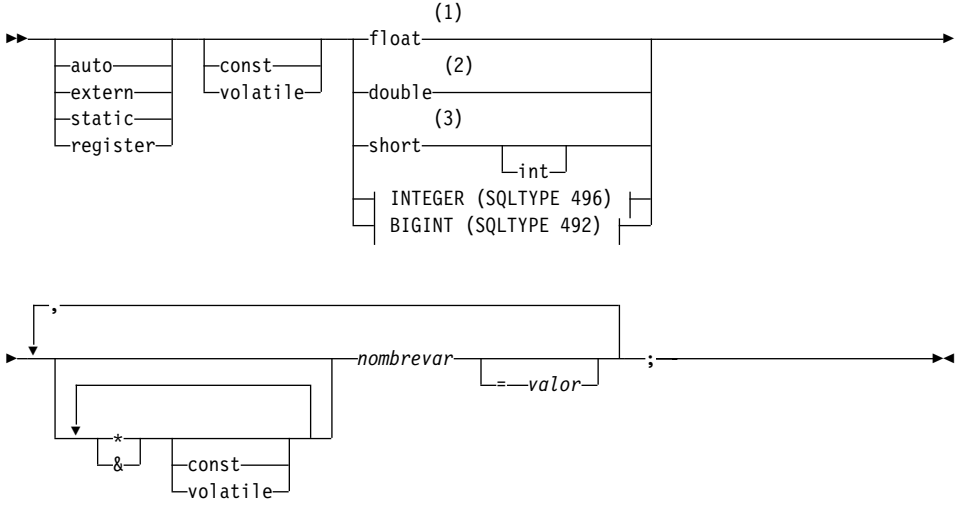
Consulta relacionada:

- “Sintaxis de las variables numéricas del lenguaje principal en C y C++” en la página 187
- “Sintaxis de las variables del lenguaje principal de tipo carácter de longitud variable en C o C++” en la página 190

Sintaxis de las variables numéricas del lenguaje principal en C y C++

A continuación se muestra la sintaxis para declarar variables numéricas del lenguaje principal en C o C++.

Sintaxis de las variables numéricas del lenguaje principal en C o C++



INTEGER (SQLTYPE 496)

BIGINT (SQLTYPE 492)

Notas:

1 REAL (SQLTYPE 480), longitud 4

2 DOUBLE (SQLTYPE 480), longitud 8

3 SMALLINT (SQLTYPE 500)

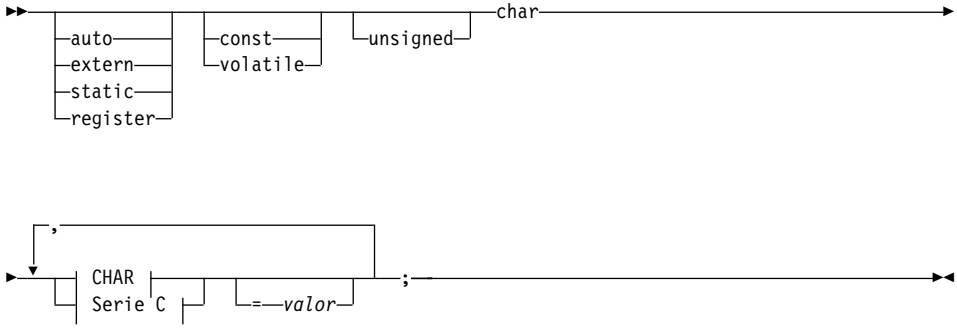
4 Para obtener una portabilidad máxima de las aplicaciones utilice sqlint32 y sqlint64 respectivamente para las variables del lenguaje principal INTEGER y BIGINT. Por omisión, el uso de variables del lenguaje principal tipo long tiene como consecuencia el error del precomiplador SQL0402 en las plataformas en que la longitud es una cantidad de 64 bits, como por ejemplo en UNIX de 64 BITS. Utilice la opción LONGERROR NO de PREP para imponer que DB2 acepte variables long como tipos de variable del lenguaje principal aceptables y las trate como variables BIGINT.

5 Para obtener una portabilidad máxima de las aplicaciones utilice, respectivamente, sqlint32 y sqlint64 para las variables del lenguaje principal INTEGER y BIGINT. Para utilizar el tipo de datos BIGINT, la plataforma tiene que soportar valores enteros de 64 bits. Por omisión, el uso de variables del lenguaje principal tipo long tiene como consecuencia el error del precomiplador SQL0402 en las plataformas en que la longitud es una cantidad de 64 bits, como por ejemplo en UNIX de 64 BITS. Utilice la opción LONGERROR NO de PREP para imponer que DB2 acepte variables long como tipos de variable del lenguaje principal aceptables y las trate como variables BIGINT.

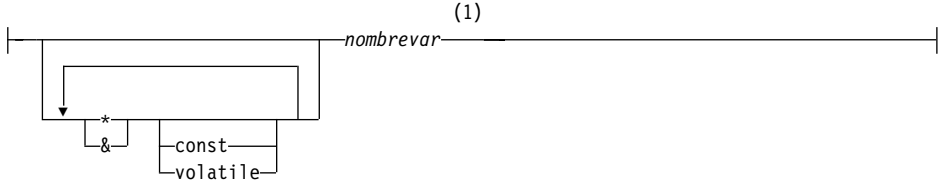
Sintaxis de las variables del lenguaje principal de tipo carácter fijas y terminadas en nulo en C y C++

A continuación se muestra la sintaxis para declarar variables del lenguaje principal de tipo carácter fijas y terminadas en nulo en C o C++.

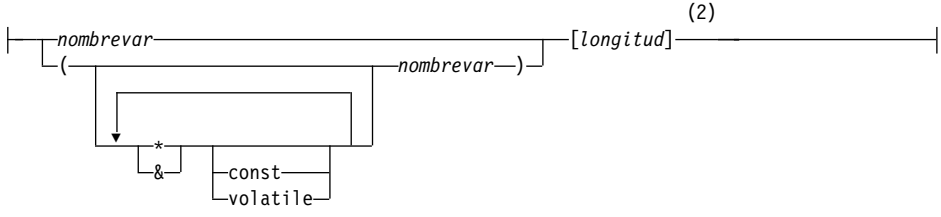
Sintaxis de las variables del lenguaje principal de tipo carácter fijas y terminadas en nulo en C o C++



CHAR



Serie C



Notas:

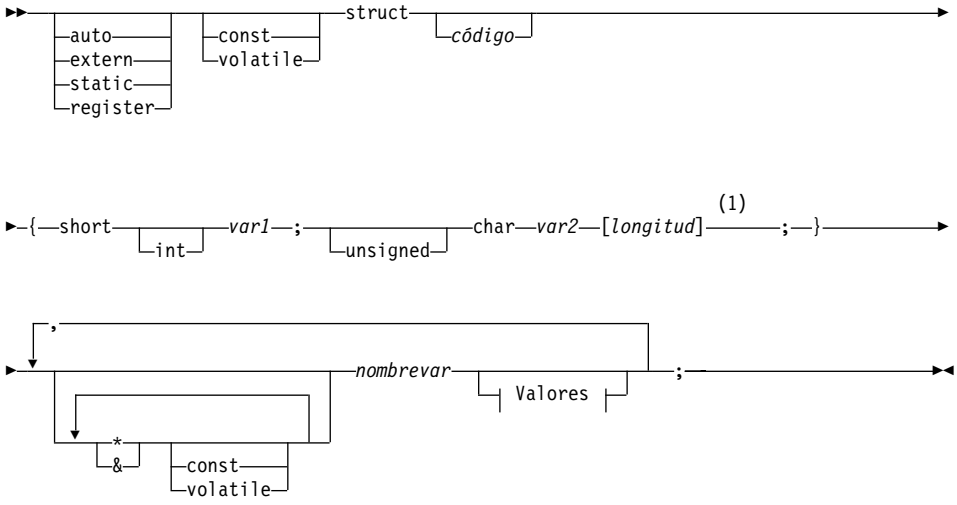
1 CHAR (SQLTYPE 452), length 1

2 Serie C terminada en nulo (SQLTYPE 460); la longitud puede ser cualquier expresión constante válida

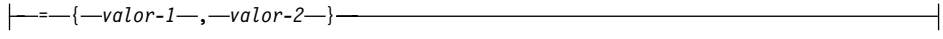
Sintaxis de las variables del lenguaje principal de tipo carácter de longitud variable en C o C++

A continuación se muestra la sintaxis para declarar variables del lenguaje principal de tipo carácter de longitud variable en C o C++.

Sintaxis para variables del lenguaje principal de tipo carácter de longitud variable en C o C++



Valores



Notas:

1 En el formato 2, la longitud puede ser cualquier expresión constante válida. Su valor después de una evaluación determina si la variable del lenguaje principal es VARCHAR (SQLTYPE 448) o LONG VARCHAR (SQLTYPE 456).

Consideraciones sobre las variables del lenguaje principal de tipo carácter de longitud variable:

1. Aunque el gestor de bases de datos convierte los datos de tipo carácter al **formato 1** o al **formato 2** siempre que es posible, el **formato 1** corresponde a los tipos de columna CHAR o VARCHAR, mientras que el **formato 2** corresponde a los tipos de columna VARCHAR y LONG VARCHAR.
2. Si se utiliza el **formato 1** con un especificador de longitud *[n]*, el valor correspondiente al especificador de longitud tras la evaluación no debe ser mayor que 32672 y la serie contenida en la variable debe terminar en nulo.

3. Si se utiliza el **formato 2**, el valor correspondiente al especificador de longitud tras la evaluación no debe ser mayor que 32700.
4. En el **formato 2**, *var1* y *var2* deben ser simples referencias a variables (y no operadores) y no se pueden utilizar como variables del lenguaje principal (*nombrevar* es la variable del lenguaje principal).
5. *nombrevar* puede ser un simple nombre de variable o puede incluir operadores, como por ejemplo **nombrevar*. Consulte la descripción de los tipos de datos de puntero en C y C++ para obtener más información.
6. El precompilador determina el SQLTYPE y la SQLLEN de todas las variables del lenguaje principal. Si una variable del lenguaje principal aparece en una sentencia de SQL con una variable de indicador, mientras dure esa sentencia se asigna como SQLTYPE el SQLTYPE base más uno.
7. El precompilador permite algunas declaraciones que no son válidas sintácticamente en C ni en C++. Si tiene dudas acerca de la sintaxis de una declaración determinada, consulte la documentación del compilador.

Conceptos relacionados:

- "Variables del lenguaje principal utilizadas como tipos de datos de puntero en C y C++" en la página 207

Variables de indicador en C y C++

Las variables de indicador se deben declarar con tipo de datos `short`.

Conceptos relacionados:

- "Tablas de indicadores en C y C++" en la página 203

Variables gráficas del lenguaje principal en C y C++

Para manejar datos gráficos en aplicaciones C o C++, utilice variables del lenguaje principal basadas en el tipo de datos `wchar_t` de C/C++ o en el tipo de datos `sqldbchar` proporcionado por DB2. Puede asignar estos tipos de variables del lenguaje principal a las columnas de una tabla que sean GRAPHIC, VARGRAPHIC o DBCLOB. Por ejemplo, puede actualizar o seleccionar datos DBCS de las columnas GRAPHIC o VARGRAPHIC de una tabla.

Existen tres formatos válidos para una variable gráfica del lenguaje principal:

- Formato de gráfico simple

  Las variables de gráfico simple del lenguaje principal tienen para SQLTYPE el valor 468/469, lo que equivale al tipo de datos GRAPHIC(1) de SQL.
- Formato de gráfico terminado en nulo

Terminado en nulo hace referencia a una situación en que todos los bytes del último carácter de la serie gráfica contienen ceros binarios ('\0's). Tienen para SQLTYPE el valor 400/401.

- Formato estructurado VARGRAPHIC

  Las variables de formato estructurado VARGRAPHIC del lenguaje principal tienen para SQLTYPE el valor 464/465 si su longitud está comprendida entre 1 y 16.336 bytes. Tienen para SQLTYPE el valor 472/473 si su longitud está comprendida entre 2.000 y 16.350 bytes.

Conceptos relacionados:

- "Nombres de variables del lenguaje principal en C y C++" en la página 185
- "Declaraciones de variables del lenguaje principal en C y C++" en la página 186
- "Inicialización de variables del lenguaje principal en C y C++" en la página 200
- "Soporte de estructuras del lenguaje principal en C y C++" en la página 201
- "Tablas de indicadores en C y C++" en la página 203
- "Codificación de caracteres de varios bytes en C y C++" en la página 210
- "Tipos de datos wchar_t y sqldbchar en C y C++" en la página 211
- "Opción del precompilador WCHARTYPE en C y C++" en la página 211

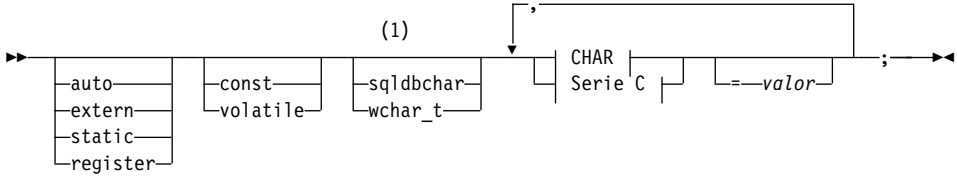
Consulta relacionada:

- "Sintaxis de la declaración gráfica de formatos gráficos de un solo gráfico y terminados en nulo en C y C++" en la página 192
- "Sintaxis para la declaración gráfica del formato estructurado VARGRAPHIC en C o C++" en la página 194
- "Sintaxis de las variables del lenguaje principal de objeto grande (LOB) en C o C++" en la página 195
- "Sintaxis de las variables del lenguaje principal de localizador de objeto grande (LOB) en C o C++" en la página 198
- "Sintaxis de declaraciones de variables del lenguaje principal de referencia de archivos en C o C++" en la página 199

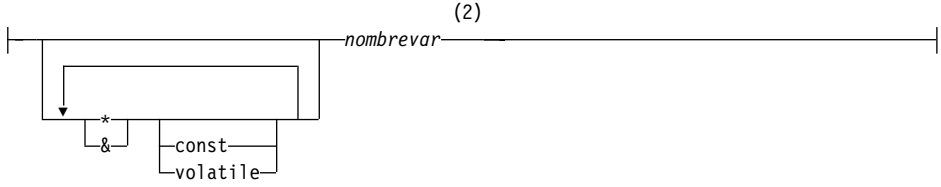
Sintaxis de la declaración gráfica de formatos gráficos de un solo gráfico y terminados en nulo en C y C++

A continuación se muestra la sintaxis para declarar una variable del lenguaje principal gráfica mediante el formato de un solo gráfico y el formato de gráfico terminado en nulo.

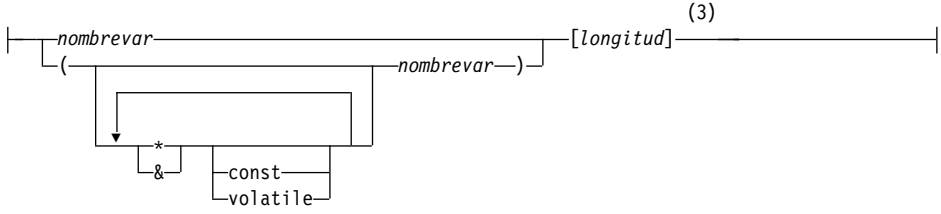
Sintaxis para la declaración gráfica de formato de un solo gráfico y formato gráfico terminado en nulo



CHAR



Serie C



Notas:

1 Para determinar cuál de los dos tipos gráficos debe utilizar, consulte la descripción de los tipos de datos wchar_t y sqldbchar en C y C++.

2 GRAPHIC (SQLTYPE 468), longitud 1

3 Serie gráfica terminada en nulo (SQLTYPE 400)

Consideraciones sobre las variables del lenguaje principal de gráficos:

1. El formato de gráfico simple declara una variable del lenguaje principal de serie gráfica y longitud fija de longitud 1 con un SQLTYPE de 468 ó 469.
2. *valor* es un inicializador. Se debe utilizar un literal de serie de caracteres amplios (literal L) si se utiliza la opción WCHARTYPE CONVERT del precompilador.
3. *longitud* puede ser cualquier expresión constante válida, y su valor después de una evaluación tiene que ser mayor o igual a 1 y no mayor que la longitud máxima de VARGRAPHIC, que es 16336.

4. Las series gráficas terminadas en nulo se manejan de forma distinta en función del valor del establecimiento de opciones de precompilación de nivel estándar.

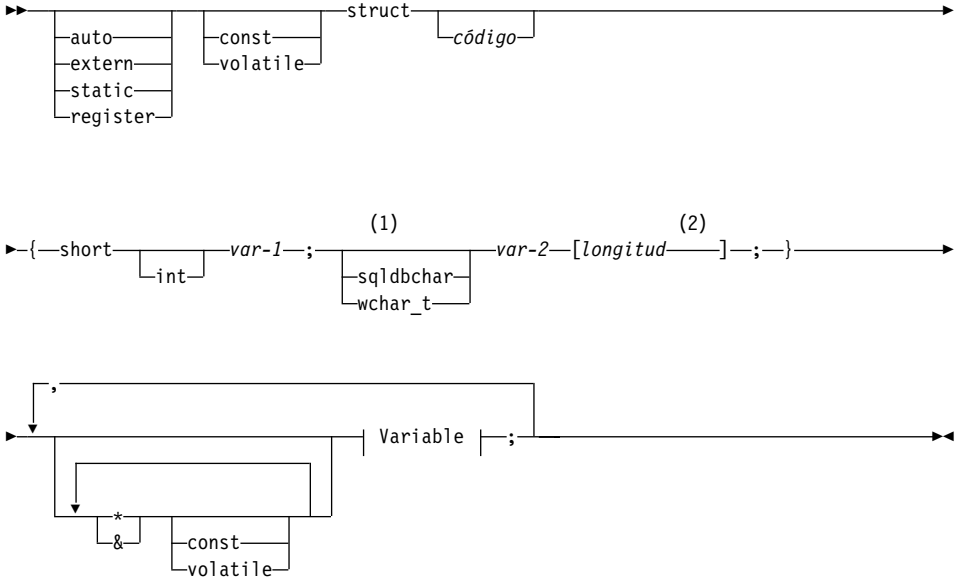
Conceptos relacionados:

- "Series terminadas en nulo en C y C++" en la página 205
- "Tipos de datos wchar_t y sqldbchar en C y C++" en la página 211

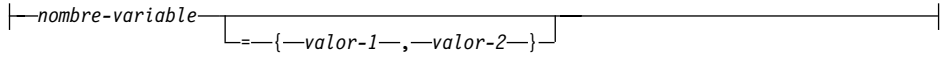
Sintaxis para la declaración gráfica del formato estructurado VARGRAPHIC en C o C++

A continuación se muestra la sintaxis para declarar una variable del lenguaje principal gráfica mediante el formato estructurado VARGRAPHIC.

Sintaxis para la declaración gráfica del formato estructurado VARGRAPHIC



Variable:



Notas:

1 Para determinar cuál de los dos tipos gráficos debe utilizar, consulte la descripción de los tipos de datos wchar_t y sqldbchar en C y C++.

2 *longitud* puede ser cualquier expresión de constante válida. Su valor después de una evaluación determina si la variable del lenguaje

principal es VARGRAPHIC (SQLTYPE 464) o LONG VARGRAPHIC (SQLTYPE 472). El valor de longitud debe ser mayor o igual que 1 y menor o igual que la longitud máxima de LONG VARGRAPHIC, que es 16350.

Consideraciones sobre la declaración gráfica (formato estructurado VARGRAPHIC):

1. *var-1* y *var-2* deben ser simples referencias a variables (y no operadores) y no se pueden utilizar como variables del lenguaje principal.
2. *valor-1* y *valor-2* son inicializadores de *var-1* y *var-2*.*valor-1* debe ser un entero y *valor-2* debe ser un literal de serie de caracteres amplios (literal L) si se utiliza la opción WCHARTYPE CONVERT del precompilador.
3. Se puede utilizar el *código* struct para definir otras áreas de datos, pero no se puede utilizar por sí mismo como variable del lenguaje principal.

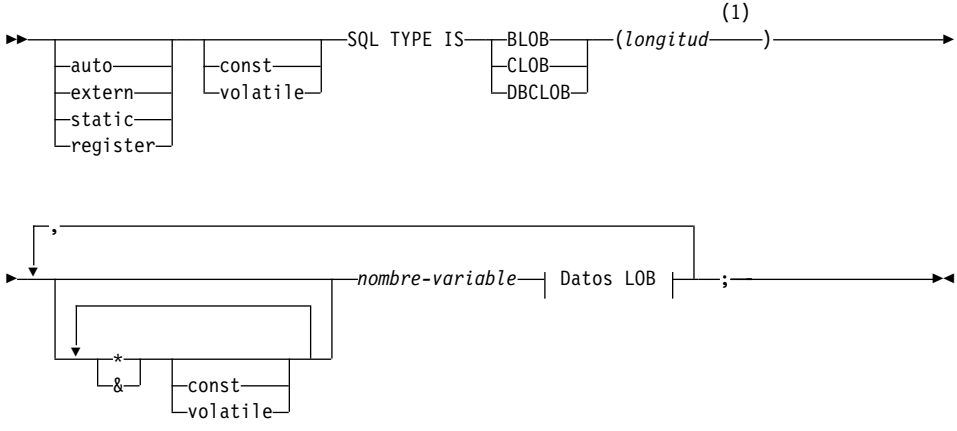
Conceptos relacionados:

- "Tipos de datos wchar_t y sqldbchar en C y C++" en la página 211

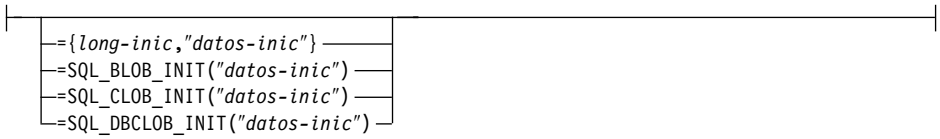
Sintaxis de las variables del lenguaje principal de objeto grande (LOB) en C o C++

A continuación se muestra la sintaxis para declarar variables del lenguaje principal de objeto grande (LOB) en C o C++.

Sintaxis de las variables del lenguaje principal de objeto grande (LOB) en C o C++



Datos LOB



Notas:

1 *longitud* puede ser cualquier expresión de constante válida en la que se pueden utilizar las constantes K, M o G. El valor de la longitud después de una evaluación para BLOB y CLOB debe ser 1 <= longitud <= 2.147.483.647. El valor de *longitud* tras la evaluación correspondiente a DBCLOB debe ser 1 <= longitud <= 1.073.741.823.

Consideraciones sobre las variables del lenguaje principal LOB:

1. Se necesita la cláusula SQL TYPE IS para poder distinguir los tres tipos de LOB entre sí, de forma que se puedan llevar a cabo una comprobación del tipo y una resolución de la función para las variables del lenguaje principal de tipo LOB que se pasan a las funciones.
2. SQL TYPE IS, BLOB, CLOB, DBCLOB, K, M, G se pueden especificar en una mezcla de mayúsculas y minúsculas.
3. La longitud máxima permitida para la serie de inicialización *"datos-inic"* es 32702 bytes, incluidos los delimitadores de la serie (igual al límite existente en series C/C++ dentro del precompilador).
4. La longitud de inicialización, *long-inic*, tiene que ser una constante numérica (esto es, no puede incluir K, M ni G).
5. Se debe especificar una longitud para el LOB; es decir, que la declaración siguiente no está permitida:

   ```
   SQL TYPE IS BLOB my_blob;
   ```

6. Si el LOB no se inicializa dentro de la declaración, no se realizará ninguna inicialización en el código generado por el precompilador.
7. Si se inicializa un DBCLOB, es responsabilidad del usuario añadirle a la serie el prefijo 'L' (que indica una serie de caracteres amplios).

   Nota: los literales de caracteres amplios, por ejemplo `L"Hello"`, sólo se deben utilizar en un programa precompilado si se selecciona la opción WCHARTYPE CONVERT de precompilación.
8. El precompilador genera un código de estructura que se puede utilizar para pasarlo al tipo de la variable del lenguaje principal.

Ejemplo de BLOB:

La declaración:

```
static Sql Type is Blob(2M) my_blob=SQL_BLOB_INIT("mydata");
```

Tiene como resultado la generación de la estructura siguiente:

```
static struct my_blob_t {
      sqluint32        length;
      char             data[2097152];
 } my_blob=SQL_BLOB_INIT("mydata");
```

Ejemplo de CLOB:

La declaración:

```
volatile sql type is clob(125m) *var1, var2 = {10, "data5data5"};
```

Tiene como resultado la generación de la estructura siguiente:

```
volatile struct var1_t {
     sqluint32        length;
     char             data[131072000];
} * var1, var2 = {10, "data5data5"};
```

Ejemplo de DBCLOB:

La declaración:

```
SQL TYPE IS DBCLOB(30000) my_dbclob1;
```

Si se precompila con la opción WCHARTYPE NOCONVERT, tiene como resultado la generación de la estructura siguiente:

```
struct my_dbclob1_t {
     sqluint32        length;
     sqldbchar        data[30000];
} my_dbclob1;
```

La declaración:

```
SQL TYPE IS DBCLOB(30000) my_dbclob2 = SQL_DBCLOB_INIT(L"mydbdata");
```

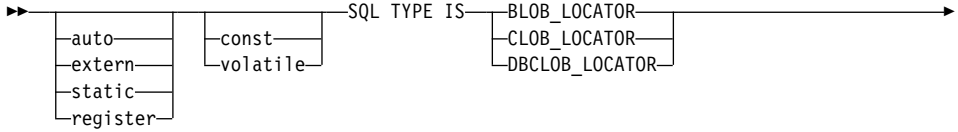
Si se precompila con la opción WCHARTYPE CONVERT, tiene como resultado la generación de la estructura siguiente:

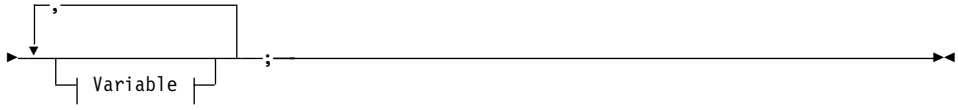
```
struct my_dbclob2_t {
     sqluint32        length;
     wchar_t          data[30000];
} my_dbclob2 = SQL_DBCLOB_INIT(L"mydbdata");
```

Sintaxis de las variables del lenguaje principal de localizador de objeto grande (LOB) en C o C++

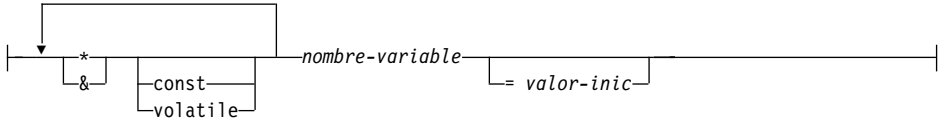
A continuación se muestra la sintaxis para declarar variables del lenguaje principal de localizador de objeto grande (LOB) en C o C++.

Sintaxis de las variables del lenguaje principal de localizador de objeto grande (LOB) en C o C++





Variable



Consideraciones sobre las variables del lenguaje principal de localizador de LOB:

1. SQL TYPE IS, BLOB_LOCATOR, CLOB_LOCATOR, DBCLOB_LOCATOR se pueden especificar en una mezcla de mayúsculas y minúsculas.
2. *valor-inic* permite la inicialización de variables de localizador de puntero y referencia. Otros tipos de inicialización no tendrán ningún significado.

Ejemplo de localizador de CLOB (existen otras declaraciones de tipos de localizador de LOB parecidas):

La declaración:

```
SQL TYPE IS CLOB_LOCATOR my_locator;
```

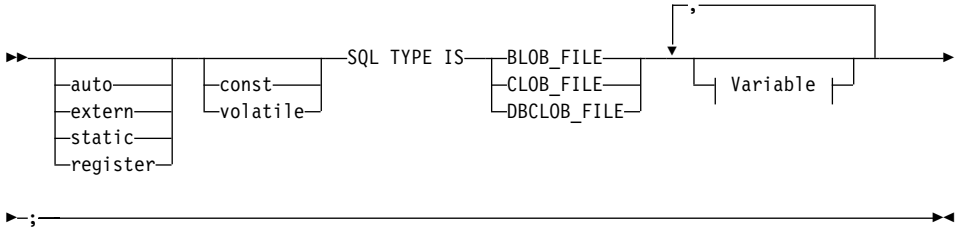
Tiene como resultado la generación de la declaración siguiente:

```
sqlint32 my_locator;
```

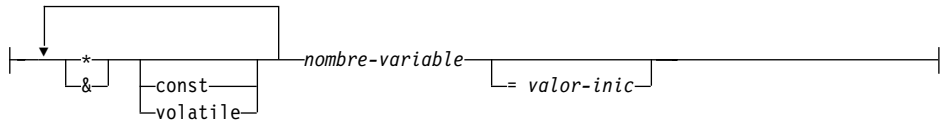
Sintaxis de declaraciones de variables del lenguaje principal de referencia de archivos en C o C++

A continuación se muestra la sintaxis para declarar variables del lenguaje principal de referencia de archivos en C o C++.

Sintaxis de las variables del lenguaje principal de referencia de archivos en C o C++



Variable



Nota: SQL TYPE IS, BLOB_FILE, CLOB_FILE, DBCLOB_FILE se pueden especificar en una mezcla de mayúsculas y minúsculas.

Ejemplo de referencia de archivos CLOB (existen otras declaraciones de tipo de referencia de archivos LOB parecidas):

La declaración:

```
static volatile SQL TYPE IS BLOB_FILE my_file;
```

Tiene como resultado la generación de la estructura siguiente:

```
static volatile struct {
     sqluint32        name_length;
     sqluint32        data_length;
     sqluint32        file_options;
              char    name[255];
} my_file;
```

Inicialización de variables del lenguaje principal en C y C++

En las secciones declare de C++ no se pueden inicializar variables del lenguaje principal mediante paréntesis. El ejemplo siguiente muestra los métodos correcto e incorrecto de inicialización en una sección de declaración:

```
EXEC SQL BEGIN DECLARE SECTION;
  short my_short_2 = 5;       /* correcto   */
  short my_short_1(5);        /* incorrecto */
EXEC SQL END DECLARE SECTION;
```

Expansión de macros en C

El precompilador C/C++ no puede procesar directamente ninguna macro de C utilizada en una declaración dentro de una sección de declaración. En lugar de esto, en primer lugar se debe preprocesar el archivo fuente mediante un preprocesador de C externo. Para ello, especifique el mandato exacto para invocar un preprocesador de C para el precompilador mediante la opción PREPROCESSOR.

Cuando se especifica la opción PREPROCESSOR, el precompilador procesa en primer lugar todas las sentencias SQL INCLUDE, incorporando al archivo fuente el contenido de todos los archivos a los que se hace referencia en la sentencia de SQL INCLUDE. A continuación, el precompilador invoca al preprocesador de C externo, utilizando el mandato que se especifique con el archivo fuente modificado como entrada. El archivo preprocesado, del cual el precompilador siempre espera que tenga la extensión `.i`, se utiliza como el nuevo archivo fuente para el resto del proceso de precompilación.

Cualquier macro #line generada por el precompilador deja de hacer referencia al archivo fuente original, pasando a hacerla al archivo preprocesado. Para relacionar los errores del compilador con el archivo fuente original, mantenga comentarios en el archivo preprocesado. Le serán de ayuda para localizar diversas secciones de los archivos fuente originales, incluidos los archivos de cabecera. La opción para mantener comentarios suele estar disponible en los preprocesadores C y la puede incluir en el mandato que especifique mediante la opción PREPROCESSOR. No debe hacer que el preprocesador de C produzca por sí mismo como salida ninguna macro #line, puesto que se podrían mezclar incorrectamente con las generadas por el precompilador.

Notas sobre la utilización de la expansión de macros:

1. El mandato que especifique mediante la opción PREPROCESSOR debe incluir todas las opciones deseadas, pero no el nombre del archivo de entrada. Por ejemplo, para C de IBM en AIX puede utilizar la opción:

   ```
   xlC -P -DMYMACRO=1
   ```

2. El precompilador espera que el mandato genere un archivo preprocesado con la extensión `.i`. Sin embargo, no se puede utilizar la redirección para

generar el archivo preprocesado. Por ejemplo, **no se puede** utilizar la opción siguiente para generar un archivo preprocesado:

```
xlC -E > x.i
```

3. Los errores que encuentre el preprocesador de C externo se notifican en un archivo que tiene el nombre correspondiente al archivo fuente original pero con la extensión `.err`.

Por ejemplo, puede utilizar la expansión de macros en el código fuente tal como se indica a continuación:

```
#define SIZE 3

EXEC SQL BEGIN DECLARE SECTION;
  char a[SIZE+1];
  char b[(SIZE+1)*3];
  struct
  {
    short length;
    char  data[SIZE*6];
  } m;
  SQL TYPE IS BLOB(SIZE+1) x;
  SQL TYPE IS CLOB((SIZE+2)*3) y;
  SQL TYPE IS DBCLOB(SIZE*2K) z;
EXEC SQL END DECLARE SECTION;
```

Las declaraciones anteriores se resuelven de la manera siguiente después de utilizar la opción PREPROCESSOR:

```
EXEC SQL BEGIN DECLARE SECTION;
  char a[4];
  char b[12];
  struct
  {
    short length;
    char  data[18];
  } m;
  SQL TYPE IS BLOB(4) x;
  SQL TYPE IS CLOB(15) y;
  SQL TYPE IS DBCLOB(6144) z;
EXEC SQL END DECLARE SECTION;
```

Soporte de estructuras del lenguaje principal en C y C++

Con el soporte de estructuras del lenguaje principal, el precompilador C/C++ permite agrupar las variables del lenguaje principal en una sola estructura del lenguaje principal. Esta función brinda una nota taquigráfica para hacer referencia a ese mismo conjunto de variables del lenguaje principal en una sentencia de SQL. Por ejemplo, se puede utilizar la siguiente estructura del lenguaje principal para acceder a algunas de las columnas de la tabla STAFF de la base de datos SAMPLE:

```
struct tag
   {
     short id;
     struct
     {
       short length;
       char  data[10];
     } name;
     struct
     {
       short   years;
       double salary;
     } info;
   } staff_record;
```

Los campos de una estructura del lenguaje principal pueden ser de cualquiera de los tipos de variable del lenguaje principal válidos. Los tipos válidos incluyen todos los tipos numéricos, de carácter y de objetos grande. También se soportan estructuras del lenguaje principal anidadas hasta 25 niveles. En el ejemplo anterior, el campo `info` es una subestructura, mientras que el campo `name` no lo es, puesto que representa un campo VARCHAR. Se aplica el mismo principio a LONG VARCHAR, VARGRAPHIC y LONG VARGRAPHIC. Asimismo, se soportan punteros a las estructuras del lenguaje principal.

Existen dos maneras de hacer referencia a las variables del lenguaje principal agrupadas en una estructura del lenguaje principal en una sentencia de SQL:

- Se puede hacer referencia al nombre de la estructura del lenguaje principal en una sentencia de SQL.

  ```
  EXEC SQL SELECT id, name, years, salary
       INTO :staff_record
       FROM staff
       WHERE id = 10;
  ```

  El precompilador convierte la referencia a `staff_record` en una lista de todos los campos declarados en la estructura del lenguaje principal, separados por comas. Cada campo se califica con los nombres de estructura del lenguaje principal de todos los niveles, a fin de evitar conflictos de denominación con otras variables del lenguaje principal o campos. Esto es equivalente al método siguiente.

- Se puede hacer referencia a nombres de estructuras del lenguaje principal completamente calificadas en una sentencia de SQL.

  ```
  EXEC SQL SELECT id, name, years, salary
       INTO :staff_record.id, :staff_record.name,
            :staff_record.info.years, :staff_record.info.salary
       FROM staff
       WHERE id = 10;
  ```

  Las referencias a nombres de campos se deben calificar por completo aunque no existan otras variables del lenguaje principal que tengan el

mismo nombre. También se puede hacer referencia a subestructuras calificadas. En el ejemplo anterior, se puede utilizar :staff_record.info para sustituir a :staff_record.info.years, :staff_record.info.salary.

Puesto que una referencia a una estructura del lenguaje principal (primer ejemplo) es equivalente a una lista de sus campos, separados por comas, existen casos en que este tipo de referencia puede conducir a un error. Por ejemplo:

```
EXEC SQL DELETE FROM :staff_record;
```

Aquí, la sentencia DELETE espera una sola variable del lenguaje principal basada en caracteres. Si en su lugar se proporciona una estructura del lenguaje principal, la sentencia tiene como resultado un error durante la precompilación:

```
SQL0087N  La variable del lenguaje principal "staff_record" es una estructura que se
donde no se permiten referencias a estructuras.
```

Otros usos de las estructuras del lenguaje principal que pueden ocasionar que se produzca un error SQL0087N incluyen: PREPARE, EXECUTE IMMEDIATE, CALL, variables de indicador y referencias a SQLDA. Las estructuras del lenguaje principal que tengan exactamente un campo están permitidas en estas situaciones, puesto que constituyen referencias a campos individuales (segundo ejemplo).

Conceptos relacionados:

• "Tablas de indicadores en C y C++" en la página 203

Tablas de indicadores en C y C++

Una tabla de indicadores es un conjunto de variables de indicador que se utilizarán con una estructura del lenguaje principal. Se debe declarar como matriz de enteros cortos. Por ejemplo:

```
short ind_tab[10];
```

El ejemplo anterior declara una tabla de indicadores con 10 elementos. El siguiente ejemplo muestra la manera en que se puede utilizar en una sentencia de SQL:

```
EXEC SQL SELECT id, name, years, salary
     INTO :staff_record INDICATOR :ind_tab
     FROM staff
     WHERE id = 10;
```

A continuación se lista cada campo de la estructura del lenguaje principal con la variable de indicador correspondiente en la tabla:

staff_record.id	ind_tab[0]

staff_record.name	ind_tab[1]
staff_record.info.years	ind_tab[2]
staff_record.info.salary	ind_tab[3]

Nota: en una sentencia de SQL no se puede hacer referencia, de forma individual, a un elemento de una tabla de indicadores, por ejemplo `ind_tab[1]`. La palabra clave INDICATOR es opcional. No es necesario que el número de campos de estructura y de indicadores coincida; los indicadores de más no se utilizan ni tampoco los campos de más que no tienen indicadores asignados.

En lugar de una tabla de indicadores, también se puede utilizar una variable de indicador escalar para proporcionar un indicador para el primer campo de la estructura del lenguaje principal. Esto equivale a tener una tabla de indicadores con un solo elemento. Por ejemplo:

```
short scalar_ind;

EXEC SQL SELECT id, name, years, salary
          INTO :staff_record INDICATOR :scalar_ind
          FROM staff
          WHERE id = 10;
```

Si se especifica una tabla de indicadores junto con una variable del lenguaje principal en lugar de una estructura del lenguaje principal, sólo se utilizará el primer elemento de la tabla de indicadores, por ejemplo `ind_tab[0]`:

```
EXEC SQL SELECT id
          INTO :staff_record.id INDICATOR :ind_tab
          FROM staff
          WHERE id = 10;
```

Si se declara una matriz de enteros cortos en una estructura del lenguaje principal:

```
struct tag
{
  short i[2];
} test_record;
```

La matriz se expandirá en sus elementos cuando se haga referencia a `test_record` en una sentencia de SQL, haciendo que `:test_record` sea equivalente a `:test_record.i[0], :test_record.i[1]`.

Conceptos relacionados:

• “Soporte de estructuras del lenguaje principal en C y C++” en la página 201

Series terminadas en nulo en C y C++

Las series terminadas en nulo de C/C++ tienen su propio SQLTYPE (460/461 para caracteres y 468/469 para gráficos).

Las series terminadas en nulo de C/C++ se manejan de forma distinta en función del valor de la opción LANGLEVEL del precompilador. Si se especifica una variable del lenguaje principal con uno de estos SQLTYPE y la longitud declarada *n* dentro de una sentencia de SQL y el número de bytes de datos (para tipos de carácter) o de caracteres de doble byte (para tipos de gráficos) es *k*, sucede lo siguiente:

- Si la opción LANGLEVEL del mandato PREP es SAA1 (valor por omisión):

  Para la salida:

Si...	**Sucede que...**
k > *n*	Se mueven *n* caracteres a la variable del lenguaje principal de destino, SQLWARN1 se establece en 'W' y SQLCODE es 0 (SQLSTATE 01004). No se coloca ningún terminador nulo en la serie. Si se ha especificado una variable de indicador con la variable del lenguaje principal, el valor de la variable de indicador se establece en *k*.
k = *n*	Se mueven *k* caracteres a la variable del lenguaje principal de destino, SQLWARN1 se establece en 'N' y SQLCODE es 0 (SQLSTATE 01004). No se coloca ningún terminador nulo en la serie. Si se ha especificado una variable de indicador con la variable del lenguaje principal, el valor de la variable de indicador se establece en 0.
k < *n*	Se mueven *k* caracteres a la variable del lenguaje principal de destino y se coloca un carácter nulo en el carácter *k + 1*. Si se ha especificado una variable de indicador con la variable del lenguaje principal, el valor de la variable de indicador se establece en 0.

  Para la entrada:

  Cuando el gestor de bases de datos encuentre una variable del lenguaje principal de entrada que tiene uno de estos

SQLTYPE y no finaliza por un terminador nulo, supondrá que el carácter *n+1* va a contener el carácter terminador nulo.

- Si la opción LANGLEVEL del mandato PREP es MIA:

  Para la salida:

Si...	**Sucede que...**
k >= *n*	Se mueven *n - 1* caracteres a la variable del lenguaje principal de destino, SQLWARN1 se establece en 'W' y SQLCODE es 0 (SQLSTATE 01004). El carácter número *n* se establece como terminador nulo. Si se ha especificado una variable de indicador con la variable del lenguaje principal, el valor de la variable de indicador se establece en *k*.
k + 1 = *n*	Se mueven *k* caracteres a la variable del lenguaje principal de destino y se coloca el terminador nulo en el carácter *n*. Si se ha especificado una variable de indicador con la variable del lenguaje principal, el valor de la variable de indicador se establece en 0.
k + 1 < *n*	Se mueven *k* caracteres a la variable del lenguaje principal de destino, se añaden *n - k -1* blancos a la derecha, a partir del carácter *k + 1*, y luego se coloca el terminador nulo en el carácter *n*. Si se ha especificado una variable de indicador con la variable del lenguaje principal, el valor de la variable de indicador se establece en 0.

  Para la entrada:

  Cuando el gestor de bases de datos encuentre una variable del lenguaje principal de entrada que tenga uno de estos SQLTYPE y no finalice por un carácter nulo, se devolverá SQLCODE -302 (SQLSTATE 22501).

Si se especifica en cualquier otro contexto de SQL, una variable del lenguaje principal con SQLTYPE 460 y longitud *n* se trata como el tipo de datos VARCHAR con longitud *n*, tal como se ha definido anteriormente. Si se especifica en cualquier otro contexto de SQL, una variable del lenguaje principal con SQLTYPE 468 y longitud *n* se trata como el tipo de datos VARGRAPHIC con longitud *n*, tal como se ha definido anteriormente.

Variables del lenguaje principal utilizadas como tipos de datos de puntero en C y C++

Las variables del lenguaje principal se pueden declarar como punteros a tipos de datos específicos, con las restricciones siguientes:

- Si una variable del lenguaje principal se declara como puntero, no se puede declarar ninguna otra variable del lenguaje principal con el mismo nombre dentro del mismo archivo fuente. El ejemplo siguiente no está permitido:

```
char mystring[20];
char (*mystring)[20];
```

- Cuando declare un puntero a una matriz de caracteres terminada en nulo, utilice paréntesis. En todos los casos restantes, no se permiten paréntesis. Por ejemplo:

```
EXEC SQL BEGIN DECLARE SECTION;
   char  (*arr)[10];  /* correcto  */
   char  *(arr);      /* incorrecto */
   char  *arr[10];    /* incorrecto */
 EXEC SQL END DECLARE SECTION;
```

  La primera declaración es un puntero a una matriz de caracteres de 10 bytes. Esta variable del lenguaje principal es válida. La segunda declaración no es válida. Los paréntesis no están permitidos en un puntero a un carácter. La tercera declaración es una matriz de punteros. Este tipo de datos no se soporta.

  La declaración de variable del lenguaje principal:

```
char *ptr
```

  se acepta, pero no significa *serie de caracteres terminada en nulo de longitud indeterminada*; significa *puntero a una variable del lenguaje principal de un solo carácter y de longitud fija*. Es posible que esto no fuera lo que se pretendía. Para definir una variable del lenguaje principal de puntero que pueda indicar distintas series de caracteres, utilice el primero de los formatos de declaración anteriores.

- Cuando se utilizan variables del lenguaje principal de puntero en sentencias de SQL, deben tener como prefijo el mismo número de asteriscos con que se han declarado, tal como en el ejemplo siguiente:

```
EXEC SQL BEGIN DECLARE SECTION;
  char (*mychar)[20];    /* Puntero a matriz de caracteres de 20 bytes */
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT column INTO :*mychar FROM table;   /* Correcta */
```

- Sólo se puede utilizar el asterisco como operador en un nombre de variable del lenguaje principal.

- La longitud máxima de un nombre de variable del lenguaje principal no se ve afectada por el número de asteriscos especificados, puesto que no se considera que los asteriscos formen parte del nombre.
- Siempre que utilice una variable de puntero en una sentencia de SQL, debe dejar la opción de precompilación del nivel de optimización (OPTLEVEL) en su valor por omisión, que es 0 (sin optimización). Esto significa que el gestor de bases de datos no realizará ninguna optimización del SQLDA.

Miembros de datos de clase utilizados como variables del lenguaje principal en C y C++

Puede declarar miembros de datos de clase como variables del lenguaje principal (pero no clases u objetos en sí mismos). El ejemplo siguiente ilustra el método a utilizar:

```
class STAFF
{
    private:
         EXEC SQL BEGIN DECLARE SECTION;
           char        staff_name[20];
           short int   staff_id;
           double      staff_salary;
         EXEC SQL END DECLARE SECTION;
         short       staff_in_db;
    .
    .
};
```

A los miembros de datos sólo se puede acceder directamente en sentencias de SQL mediante el puntero *this* implícito proporcionado por el compilador C++ en las funciones de miembros de clases. **No se puede** calificar explícitamente una instancia de un objeto (como, por ejemplo, `SELECT name INTO :my_obj.staff_name ...`) en una sentencia de SQL.

Si se hace una referencia directa a miembros de datos de clase en sentencias de SQL, el gestor de bases de datos resuelve la referencia utilizando el puntero *this*. Por este motivo, debe dejar la opción de precompilación del nivel de optimización (OPTLEVEL) en su valor por omisión, que es 0 (sin optimización). Esto significa que el gestor de bases de datos no realizará ninguna optimización del SQLDA. (Esto es así siempre que existan variables del lenguaje principal de puntero implicadas en las sentencias de SQL.)

El ejemplo siguiente muestra cómo se pueden utilizar directamente los miembros de datos de clase que ha declarado como variables del lenguaje principal en una sentencia de SQL.

```
class STAFF
{

:
    public:

:
        short int hire( void )
        {
          EXEC SQL INSERT INTO staff ( name,id,salary )
            VALUES ( :staff_name, :staff_id, :staff_salary );
          staff_in_db = (sqlca.sqlcode == 0);
          return sqlca.sqlcode;
        }
};
```

En este ejemplo, los miembros de datos de clase `staff_name`, `staff_id` y `staff_salary` se utilizan directamente en la sentencia INSERT. Puesto que se han declarado como variables del lenguaje principal (consulte el primer ejemplo de esta sección), están calificados de forma implícita para el objeto actual con el puntero *this*. En sentencias de SQL también se puede hacer referencia a miembros de datos a los que no se pueda acceder mediante el puntero *this*. Esto se logra haciendo una referencia indirecta a ellos mediante variables del lenguaje principal de puntero o de referencia.

El ejemplo siguiente muestra un nuevo método, *asWellPaidAs*, que toma un segundo objeto, *otherGuy*. Este método hace referencia a sus miembros indirectamente, mediante una variable del lenguaje principal de puntero local o de referencia, puesto que no se puede hacer una referencia directa a sus miembros dentro de la sentencia de SQL.

```
short int STAFF::asWellPaidAs( STAFF otherGuy )
{
    EXEC SQL BEGIN DECLARE SECTION;
      short &otherID = otherGuy.staff_id
      double otherSalary;
    EXEC SQL END DECLARE SECTION;
    EXEC SQL SELECT SALARY INTO :otherSalary
      FROM STAFF WHERE id = :otherID;
      if( sqlca.sqlcode == 0 )
         return staff_salary >= otherSalary;
      else
         return 0;
}
```

Operadores de calificación y de miembro en C y C++

No se puede utilizar el operador de resolución de ámbito de C++ '::' ni los operadores de miembros de C/C++ '.' ni '->' en sentencias de SQL incorporado. Se puede lograr fácilmente lo mismo mediante el uso de

variables de puntero o de referencia locales, que se establecen fuera de la sentencia de SQL, de forma que apunten a la variable de ámbito que se desee y luego se utilizan dentro de la sentencia de SQL para hacer referencia a ésta. El ejemplo siguiente muestra el método correcto a utilizar:

```
EXEC SQL BEGIN DECLARE SECTION;
  char (& localName)[20] = ::name;
EXEC SQL END DECLARE SECTION;
EXEC SQL
  SELECT name INTO :localName FROM STAFF
  WHERE name = 'Sanders';
```

Codificación de caracteres de varios bytes en C y C++

Algunos esquemas de codificación de caracteres, especialmente aquéllos que proceden de los países del este asiático, necesitan varios bytes para representar un carácter. Esta representación externa de los datos se denomina representación de un carácter por medio de *código de caracteres de varios bytes* e incluye los caracteres de doble byte (caracteres representados por dos bytes). Los datos gráficos en DB2 consisten en caracteres de doble byte.

Para manipular series de caracteres que contienen caracteres de doble byte puede resultar conveniente que una aplicación utilice una representación interna de los datos. Esta representación interna se denomina representación de caracteres de doble byte por medio de *código de caracteres amplios* y es el formato utilizado habitualmente en el tipo de datos `wchar_t` de C/C++. Se dispone de subrutinas que se ajustan a C de ANSI y a X/OPEN Portability Guide 4 (XPG4) para procesar los datos de caracteres amplios y para convertir los datos que se encuentran en este formato al formato de varios bytes, y viceversa.

Observe que, aunque una aplicación puede procesar datos de tipo carácter en formato de varios bytes o en formato de caracteres amplios, la interacción con el gestor de bases de datos sólo se realiza con códigos de caracteres DBCS (de varios bytes). Es decir, los datos se almacenan y se recuperan de las columnas GRAPHIC en formato DBCS. Se proporciona la opción WCHARTYPE del precompilador para permitir que los datos de una aplicación que están en formato de caracteres amplios se conviertan al formato de varios bytes, y viceversa, cuando se produce el intercambio de datos con el motor de la base de datos.

Conceptos relacionados:

- “Variables gráficas del lenguaje principal en C y C++” en la página 191
- “Tipos de datos wchar_t y sqldbchar en C y C++” en la página 211

Tipos de datos wchar_t y sqldbchar en C y C++

Mientras que el tamaño y la codificación de datos gráficos de DB2 son constantes en todas las plataformas para una página de códigos determinada, el tamaño y el formato interno del tipo de datos `wchar_t` de C o de C++ de ANSI dependen del compilador que se utilice y de la plataforma en que se encuentre. No obstante, DB2 define el tipo de datos `sqldbchar` con un tamaño de dos bytes, y se pretende que constituya una manera portátil de manipular los datos DBCS y UCS-2 en el mismo formato en que están almacenados en la base de datos.

Puede definir todos los tipos de variables del lenguaje principal de gráficos C de DB2 mediante `wchar_t` o `sqldbchar`. Debe utilizar `wchar_t` si crea la aplicación mediante la opción de precompilación WCHARTYPE CONVERT.

Nota: cuando especifique la opción WCHARTYPE CONVERT en una plataforma Windows, tenga en cuenta que `wchar_t` en plataformas Windows es Unicode. Por lo tanto, si el `wchar_t` del compilador C/C++ no es Unicode, es posible que la llamada a la función `wcstombs()` falle con SQLCODE -1421 (SQLSTATE=22504). Si esto sucede, puede especificar la opción WCHARTYPE NOCONVERT y llamar a las funciones `wcstombs()` y `mbstowcs()` de forma explícita desde dentro del programa.

Si crea la aplicación con la opción de precompilación WCHARTYPE NOCONVERT, debe utilizar `sqldbchar` para conseguir una portabilidad máxima entre distintas plataformas de cliente y servidor DB2. Puede utilizar `wchar_t` con WCHARTYPE NOCONVERT, pero sólo en aquellas plataformas en que `wchar_t` esté definido con una longitud de dos bytes.

Si utiliza incorrectamente `wchar_t` o `sqldbchar` en declaraciones de variables del lenguaje principal, durante la precompilación recibirá un SQLCODE 15 (sin SQLSTATE).

Conceptos relacionados:

- “Opción del precompilador WCHARTYPE en C y C++” en la página 211
- “Consideraciones sobre los juegos de códigos EUC y UCS-2 de japonés y chino tradicional” en la página 444

Opción del precompilador WCHARTYPE en C y C++

Mediante la opción WCHARTYPE del precompilador, puede especificar el formato de caracteres gráficos que desea utilizar en la aplicación C/C++. Esta opción proporciona flexibilidad para elegir entre tener los datos gráficos en formato de varios bytes o en formato de caracteres amplios. La opción WCHARTYPE tiene dos valores posibles:

CONVERT

Si se selecciona la opción WCHARTYPE CONVERT, se convierten los códigos de caracteres entre la variable de gráficos del lenguaje principal y el gestor de bases de datos. Para las variables del lenguaje principal de entrada de gráficos, la conversión de código de caracteres del formato de caracteres amplios a formato de caracteres DBCS de varios bytes se realiza antes de enviar los datos al gestor de bases de datos, mediante la función `wcstombs()` de C de ANSI. Para las variables del lenguaje principal de salida de gráficos, la conversión de código de caracteres del formato de caracteres DBCS de varios bytes al formato de caracteres amplios se realiza antes de que los datos recibidos del gestor de bases de datos se almacenen en la variable del lenguaje principal, mediante la función `mbstowcs()` de C de ANSI.

La ventaja de utilizar WCHARTYPE CONVERT es que permite que la aplicación aproveche plenamente los mecanismos de C de ANSI para tratar las series de caracteres amplios (literales L, funciones de serie 'wc', etc.) sin tener que convertir los datos de forma explícita al formato de varios bytes antes de establecer comunicación con el gestor de bases de datos. El inconveniente es que las conversiones implícitas puede tener un impacto sobre el rendimiento de la aplicación durante la ejecución de ésta y pueden incrementar los requisitos de memoria.

Si selecciona WCHARTYPE CONVERT, declare todas las variables del lenguaje principal de gráficos mediante `wchar_t` en lugar de mediante `sqldbchar`.

Si desea el comportamiento de WCHARTYPE CONVERT pero no es necesario precompilar la aplicación (por ejemplo, una aplicación CLI), defina la macro `SQL_WCHART_CONVERT` del preprocesador de C durante la compilación. Así se asegurará de que determinadas definiciones de los archivos de cabecera de DB2 utilicen el tipo de datos `wchar_t` en lugar de `sqldbchar`.

Nota: la opción de precompilación WCHARTYPE CONVERT no recibe soporte actualmente en programas que se ejecutan en el cliente DB2 Windows 3.1. Para estos programas, utilice el valor por omisión (WCHARTYPE NOCONVERT).

NOCONVERT (valor por omisión)

Si se elige la opción WCHARTYPE NOCONVERT o no se especifica ninguna opción WCHARTYPE, no se produce ninguna conversión implícita del código de caracteres entre la aplicación y el gestor de bases de datos. Los datos de una variable del lenguaje principal de gráficos se envían al gestor de bases de datos y se reciben del mismo como caracteres DBCS inalterados. Esto tiene la ventaja de que se mejora el rendimiento, pero el inconveniente de que la aplicación debe abstenerse de utilizar datos de caracteres amplios en las variables del

lenguaje principal wchar_t, o tiene que llamar explícitamente a las funciones wcstombs() y mbstowcs() para convertir los datos al formato de varios bytes, o viceversa, cuando interactúe con el gestor de bases de datos.

Si selecciona WCHARTYPE NOCONVERT, declare todas las variables del lenguaje principal de gráficos utilizando el tipo sqldbchar, a fin de obtener la máxima portabilidad a otras plataformas cliente/servidor de DB2.

Hay otras directrices que debe tener en cuenta, que son las siguientes:

- Puesto que se utiliza el soporte de wchar_t o sqldbchar para manejar datos DBCS, su uso requiere que el hardware y el software tengan capacidad de DBCS o EUC. Este soporte sólo está disponible en el entorno DBCS de DB2 Universal Database, o para tratar datos GRAPHIC en cualquier aplicación (incluidas las aplicaciones de un solo byte) conectada a una base de datos UCS-2.
- Los caracteres que no sean DBCS, y los caracteres amplios que se pueden convertir en caracteres no DBCS, no se deben utilizar en series gráficas. *Caracteres que no sean DBCS* hace referencia a los caracteres de un solo byte y a los caracteres que no son de doble byte. Las series gráficas no se validan para asegurar que sus valores únicamente contienen puntos de código de caracteres de doble byte. Las variables del lenguaje principal de gráficos sólo deben contener datos DBCS o, si WCHARTYPE CONVERT está en vigor, datos de caracteres amplios que se convierten en datos DBCS. Debe almacenar los datos que contienen una mezcla de caracteres de doble byte y de un solo byte en variables del lenguaje principal de tipo carácter. Observe que las variables del lenguaje principal de datos mixtos no se ven afectadas por el establecimiento de la opción WCHARTYPE.
- En las aplicaciones en que se utiliza la opción de precompilación WCHARTYPE NOCONVERT, no se deben utilizar literales L junto con variables del lenguaje principal de gráficos, puesto que los literales L están en formato de caracteres amplios. Un literal L es una serie de caracteres amplios C que tiene como prefijo la letra L y como tipo de datos "array of wchar_t". Por ejemplo, L"dbcs-string" es un literal L.
- En las aplicaciones en que se utiliza la opción de precompilación WCHARTYPE CONVERT, se pueden utilizar literales L para inicializar variables del lenguaje principal wchar_t, pero no se pueden utilizar en sentencias de SQL. En lugar de utilizar literales L, las sentencias de SQL deben usar constantes de serie de gráficos, que son independientes del valor de WCHARTYPE.
- El valor de la opción WCHARTYPE afecta a los datos de gráficos que se pasan al gestor de bases de datos y que se reciben de éste utilizando la estructura SQLDA, así como a las variables del lenguaje principal. Si WCHARTYPE CONVERT está en vigor, se supondrá que los datos gráficos

recibidos de la aplicación mediante una SQLDA están en formato de caracteres amplios, y se convertirán al formato DBCS a través de una llamada implícita a `wcstombs()`. De forma parecida, los datos de salida de gráficos recibidos por una aplicación se habrán convertido al formato de caracteres amplios antes de colocarlos en el almacenamiento de la aplicación.

- Los procedimientos almacenados sin barrera se deben precompilar con la opción WCHARTYPE NOCONVERT. Los procedimientos almacenados normales con barrera se pueden precompilar con las opciones CONVERT o NOCONVERT, lo cual afectará al formato de los datos gráficos manipulados por las sentencias de SQL contenidas en el procedimiento almacenado. No obstante, y en cualquier caso, los datos gráficos que se pasen al procedimiento almacenado mediante el SQLDA estarán en formato DBCS. De la misma forma, los datos que se pasen desde el procedimiento almacenado mediante el SQLDA deben estar en formato DBCS.
- Si una aplicación llama a un procedimiento almacenado a través de la interfaz Database Application Remote Interface (DARI) (la API `sqleproc()`), los datos gráficos del SQLDA de entrada tienen que estar en formato DBCS, o en UCS-2 si está conectada a una base de datos UCS-2, independientemente del estado del valor de WCHARTYPE de la aplicación que realiza la llamada. Asimismo, los datos gráficos del SQLDA de salida se devolverán en formato DBCS, o en UCS-2 si está conectada a una base de datos UCS-2, independientemente del valor de WCHARTYPE.
- Si una aplicación llama a un procedimiento almacenado mediante la sentencia CALL de SQL, se producirá una conversión de los datos gráficos en el SQLDA, según el valor de WCHARTYPE de la aplicación que realiza la llamada.
- Los datos gráficos que se pasen a funciones definidas por el usuario (UDF) siempre estarán en formato DBCS. De la misma forma, se supondrá que los datos gráficos devueltos desde una UDF están en formato DBCS para las bases de datos DBCS, y en formato UCS-2 para las bases de datos EUC y UCS-2.
- Los datos almacenados en archivos DBCLOB mediante el uso de variables de referencia a archivos DBCLOB se almacenan en formato DBCS o, en el caso de bases de datos UCS-2, en formato UCS-2. Del mismo modo, los datos de entrada procedentes de archivos DBCLOB se recuperan en formato DBCS o, en el caso de bases de datos UCS-2, en formato UCS-2.

Nota: si se precompilan aplicaciones C utilizando la opción WCHARTYPE CONVERT, DB2 valida los datos gráficos de la aplicación tanto de entrada como de salida, puesto que los datos se pasan a través de las funciones de conversión. Si *no* se utiliza la opción CONVERT, no se produce ninguna conversión de los datos gráficos, por lo que tampoco se produce ninguna validación. En un entorno mixto de CONVERT/NOCONVERT, se pueden ocasionar problemas si una

aplicación NOCONVERT inserta datos gráficos no válidos que luego son captados por una aplicación CONVERT. Estos datos no se pueden convertir con SQLCODE -1421 (SQLSTATE 22504) en una operación FETCH de la aplicación CONVERT.

Consulta relacionada:

- "PREPARE sentencia" en el manual *Consulta de SQL, Volumen 2*

Consideraciones sobre EUC en japonés o chino tradicional y UCS-2 en C y C++

Si la página de códigos de la aplicación es EUC en japonés o chino tradicional, o si la aplicación conecta con una base de datos UCS-2, puede acceder a columnas GRAPHIC del servidor de base de datos utilizando las opciones CONVERT o NOCONVERT y las variables del lenguaje principal de gráficos `wchar_t` o `sqldbchar`, o las SQLDA de entrada/salida. En este apartado, *formato DBCS* hace referencia al esquema de codificación de UCS-2 para datos EUC. Considere los casos siguientes:

- Se utiliza la opción CONVERT

  El cliente DB2 convierte los datos gráficos del formato de caracteres amplios a la página de códigos de la aplicación, y luego a UCS-2, antes de enviar el SQLDA de entrada al servidor de base de datos. Los datos gráficos se envían al servidor de base de datos identificados por el identificador de página de códigos de UCS-2. Los datos de tipo carácter mixtos se identifican con el identificador de página de códigos de la aplicación. Cuando un cliente recupera datos gráficos de una base de datos, éstos se identifican con el identificador de página de códigos de UCS-2. El cliente DB2 convierte los datos de UCS-2 a la página de códigos de la aplicación cliente y luego al formato de caracteres amplios. Si se utiliza una SQLDA de entrada en lugar de una variable del lenguaje principal, se requiere al usuario que se asegure de que los datos gráficos se han codificado utilizando el formato de caracteres amplios. Estos datos se convertirán a UCS-2 y luego se enviarán al servidor de base de datos. Estas conversiones tendrán impacto en el rendimiento.

- Se utiliza la opción NOCONVERT

  DB2 supone que los datos se han codificado utilizando UCS-2 y están identificados por la página de códigos de UCS-2, y no se produce ninguna conversión. DB2 supone que la variable del lenguaje principal de gráficos se utiliza simplemente como contenedor. Si no se selecciona la opción NOCONVERT, los datos gráficos recuperados del servidor de base de datos se pasan a la aplicación codificados mediante UCS-2. Las posibles conversiones de la página de códigos de la aplicación a UCS-2 y de UCS-2 a la página de códigos de la aplicación son responsabilidad del usuario. Los datos identificados como UCS-2 se envían al servidor de base de datos sin que se produzca ninguna conversión ni alteración.

Para minimizar las conversiones, puede utilizar la opción NOCONVERT y manejar las conversiones en la aplicación, o bien no utilizar columnas GRAPHIC. Para los entornos de cliente en que la codificación `wchar_t` es en Unicode de dos bytes, por ejemplo en Windows NT o AIX versión 4.3 y posteriores, puede utilizar la opción NOCONVERT y trabajar directamente con UCS-2. En estos casos, la aplicación debe manejar la diferencia entre las arquitecturas big-endian y little-endian. Con la opción NOCONVERT, DB2 Universal Database utiliza `sqldbchar`, que es siempre un big-endian de dos bytes.

No asigne datos IBM-eucJP / IBM-eucTW CS0 (ASCII de 7 bits) e IBM-eucJP CS2 (Katakana) a variables del lenguaje principal de gráficos después de una conversión a UCS-2 (si se especifica NOCONVERT) ni mediante una conversión al formato de caracteres amplios (si se especifica CONVERT). Esto es así porque los caracteres en estos juegos de códigos EUC pasan a tener un solo byte cuando se convierten de UCS-2 a DBCS de PC.

En general, aunque eucJP y eucTW almacenan los datos gráficos (GRAPHIC) como UCS-2, los datos GRAPHIC contenidos en estas bases de datos siguen siendo datos eucJP o eucTW no ASCII. Concretamente, cualquier espacio rellenado en estos datos GRAPHIC es espacio DBCS (también conocido como espacio ideográfico en UCS-2, U+3000). Sin embargo, para una base de datos UCS-2, los datos GRAPHIC pueden contener cualquier carácter UCS-2 y el relleno del espacio se realiza con espacio UCS-2, U+0020. Tenga presente esta diferencia cuando codifique aplicaciones para recuperar datos UCS-2 de una base de datos UCS-2, en comparación con la recuperación de datos UCS-2 de bases de datos eucJP y eucTW.

Conceptos relacionados:

- “Consideraciones sobre los juegos de códigos EUC y UCS-2 de japonés y chino tradicional” en la página 444

Sección declare de SQL con variables del lenguaje principal para C y C++

A continuación se muestra una sección de declaración de SQL de ejemplo con variables del lenguaje principal declaradas para tipos de datos SQL soportados:

```
EXEC SQL BEGIN DECLARE SECTION;

⋮
        short     age = 26;              /* tipo de SQL  500 */
        short     year;                  /* tipo de SQL  500 */
        sqlint32  salary;                /* tipo de SQL  496 */
        sqlint32  deptno;                /* tipo de SQL  496 */
        float     bonus;                 /* tipo de SQL  480 */
        double    wage;                  /* tipo de SQL  480 */
```

```
char      mi;                    /* tipo de SQL  452 */
char      name[6];               /* tipo de SQL  460 */
struct    {
          short len;
          char data[24];
          } address;             /* tipo de SQL  448 */
struct    {
          short len;
          char data[32695];
          } voice;               /* tipo de SQL  456 */
sql type is clob(1m)
          chapter;               /* tipo de SQL  408 */
sql type is clob_locator
          chapter_locator;       /* tipo de SQL  964 */
sql type is clob_file
          chapter_file_ref;      /* tipo de SQL  920 */
sql type is blob(1m)
          video;                 /* tipo de SQL  404 */
sql type is blob_locator
          video_locator;         /* tipo de SQL  960 */
sql type is blob_file
          video_file_ref;        /* tipo de SQL  916 */
sql type is dbclob(1m)
          tokyo_phone_dir;       /* tipo de SQL  412 */
sql type is dbclob_locator
          tokyo_phone_dir_lctr;  /* tipo de SQL  968 */
sql type is dbclob_file
          tokyo_phone_dir_flref; /* tipo de SQL  924 */
struct    {
          short len;
          sqldbchar data[100];
          } vargraphic1;         /* tipo de SQL  464 */
                                 /* Precompilado con la
                                 opción WCHARTYPE NOCONVERT */
struct    {
          short len;
          wchar_t data[100];
          } vargraphic2;         /* tipo de SQL  464 */
                                 /* Precompilado con la
                                 opción WCHARTYPE CONVERT */
struct    {
          short len;
          sqldbchar data[10000];
          } long_vargraphic1;    /* tipo de SQL  472 */
                                 /* Precompilado con la
                                 opción WCHARTYPE NOCONVERT */
struct    {
          short len;
          wchar_t data[10000];
          } long_vargraphic2;    /* tipo de SQL  472 */
                                 /* Precompilado con la
                                 opción WCHARTYPE CONVERT */
sqldbchar graphic1[100];         /* tipo de SQL  468 */
                                 /* Precompilado con la
                                 opción WCHARTYPE NOCONVERT */
```

```
        wchar_t   graphic2[100];          /* tipo de SQL  468 */
                                          /* Precompilado con la
                                          opción WCHARTYPE CONVERT */
        char      date[11];               /* tipo de SQL  384 */
        char      time[9];                /* tipo de SQL  388 */
        char      timestamp[27];          /* tipo de SQL  392 */
        short     wage_ind;               /* Indicador de nulo */

    ⋮
      EXEC SQL END DECLARE SECTION;
```

Consideraciones sobre tipos de datos para C y C++

Las secciones siguientes describen cómo se correlacionan los tipos de datos de SQL con tipos de datos de C y C++.

Tipos de datos SQL soportados en C y C++

Ciertos tipos de datos predefinidos de C y C++ corresponden con los tipos de columna del gestor de bases de datos. Sólo se pueden declarar como variables del lenguaje principal estos tipos de datos de C/C++.

La tabla siguiente muestra el equivalente en C/C++ de cada tipo de columna. Cuando el precompilador encuentra una declaración de una variable del lenguaje principal, determina el valor del tipo de SQL apropiado. El gestor de bases de datos utiliza este valor para convertir los datos intercambiados entre la aplicación y él mismo.

Nota: no existe soporte de variables del lenguaje principal para el tipo de datos DATALINK en ninguno de los lenguajes principales de DB2.

Tabla 13. Tipos de datos SQL correlacionados con declaraciones C/C++

Tipo de columna SQL[1]	Tipo de datos C/C++	Descripción del tipo de columna SQL
SMALLINT (500 ó 501)	short short int sqlint16	Entero con signo de 16 bits
INTEGER (496 ó 497)	long long int sqlint32[2]	Entero con signo de 32 bits
BIGINT (492 ó 493)	long long long __int64 sqlint64[3]	Entero con signo de 64 bits
REAL[4] (480 ó 481)	float	Coma flotante de precisión simple
DOUBLE[5] (480 ó 481)	double	Coma flotante de precisión doble

Tabla 13. Tipos de datos SQL correlacionados con declaraciones C/C++ (continuación)

Tipo de columna SQL[1]	**Tipo de datos C/C++**	**Descripción del tipo de columna SQL**
DECIMAL(*p*,*s*) (484 ó 485)	No existe un equivalente exacto; utilice double	Decimal empaquetado (Considere la posibilidad de utilizar las funciones CHAR y DECIMAL para manipular los campos decimales empaquetados como datos de tipo carácter.)
CHAR(1)(452 ó 453)	char	Carácter simple
CHAR(*n*) (452 ó 453)	No existe un equivalente exacto; utilice char[*n+1*], donde n es suficientemente grande para contener los datos 1<=*n*<=254	Serie de caracteres de longitud fija

VARCHAR(*n*)
(448 ó 449)
	struct tag {
	short int;
	char[*n*]
	}
	
	1<=*n*<=32 672
		Serie de caracteres de longitud variable no terminada en nulo con indicador de longitud de serie de 2 bytes

	Como alternativa, utilice char[*n+1*], donde n es suficientemente grande para contener los datos 1<=*n*<=32 672	Serie de caracteres de longitud variable terminada en nulo **Nota:** se le asigna un tipo SQL de 460/461.

LONG VARCHAR
(456 ó 457)
	struct tag {
	short int;
	char[*n*]
	}
	
	32 673<=*n*<=32 700
		Serie de caracteres de longitud variable no terminada en nulo con indicador de longitud de serie de 2 bytes

CLOB(*n*) (408 ó 409)	sql type is clob(*n*) 1<=*n*<=2 147 483 647	Serie de caracteres de longitud variable no terminada en nulo con indicador de longitud de serie de 4 bytes
Variable de localizador CLOB[6] (964 ó 965)	sql type is clob_locator	Identifica las entidades CLOB que residen en el servidor
Variable de referencia de archivo CLOB[6] en la página 221 (920 ó 921)	sql type is clob_file	Descriptor del archivo que contiene datos CLOB
BLOB(*n*) (404 ó 405)	sql type is blob(*n*) 1<=*n*<=2 147 483 647	Serie binaria variable no terminada en nulo con indicador de longitud de serie de 4 bytes
Variable de localizador BLOB[6] (960 ó 961)	sql type is blob_locator	Identifica las entidades BLOB del servidor
Variable de referencia de archivo BLOB[6] (916 ó 917)	sql type is blob_file	Descriptor del archivo que contiene datos BLOB

Tabla 13. Tipos de datos SQL correlacionados con declaraciones C/C++ (continuación)

Tipo de columna SQL[1]	**Tipo de datos C/C++**	**Descripción del tipo de columna SQL**
DATE (384 ó 385)	Formato de caracteres terminado en nulo	Admitir como mínimo 11 caracteres para acomodar el terminador nulo.
	Formato estructurado VARCHAR	Admitir como mínimo 10 caracteres.
TIME (388 ó 389)	Formato de caracteres terminado en nulo	Admitir como mínimo 9 caracteres para acomodar el terminador nulo.
	Formato estructurado VARCHAR	Admitir como mínimo 8 caracteres.
TIMESTAMP (392 ó 393)	Formato de caracteres terminado en nulo	Admitir como mínimo 27 caracteres para acomodar el terminador nulo.
	Formato estructurado VARCHAR	Admitir como mínimo 26 caracteres.
Nota: los tipos de datos siguientes sólo están disponibles en los entornos DBCS o EUC si se compilan previamente con la opción WCHARTYPE NOCONVERT.		
GRAPHIC(1) (468 ó 469)	sqldbchar	Carácter simple de doble byte
GRAPHIC(*n*) (468 ó 469)	No existe un equivalente exacto; utilice sqldbchar[*n+1*], donde n es suficientemente grande para contener los datos 1<=*n*<=127	Serie de caracteres de doble byte y longitud fija

VARGRAPHIC(*n*) (464 ó 465)	struct tag {
	short int;
	sqldbchar[*n*]
	}
	
	1<=*n*<=16 336
		Serie de caracteres de doble byte y longitud variable no terminada en nulo con indicador de longitud de serie de 2 bytes

	Como alternativa, utilice sqldbchar[*n+1*], donde n es suficientemente grande para contener los datos 1<=*n*<=16 336	Serie de caracteres de doble byte y longitud variable terminada en nulo **Nota:** se le asigna un tipo SQL de 400/401.

LONG VARGRAPHIC (472 ó 473)	struct tag {
	short int;
	sqldbchar[*n*]
	}
	
	16 337<=*n*<=16 350
		Serie de caracteres de doble byte y longitud variable no terminada en nulo con indicador de longitud de serie de 2 bytes

Nota: los tipos de datos siguientes sólo están disponibles en los entornos DBCS o EUC si se compilan previamente con la opción WCHARTYPE CONVERT.		
GRAPHIC(1) (468 ó 469)	wchar_t	• Carácter amplio simple (para tipo C) • Carácter de doble byte simple (para tipo de columna)
GRAPHIC(*n*) (468 ó 469)	No existe un equivalente exacto; utilice wchar_t [*n+1*], donde n es suficientemente grande para contener los datos 1<=*n*<=127	Serie de caracteres de doble byte y longitud fija

Tabla 13. Tipos de datos SQL correlacionados con declaraciones C/C++ (continuación)

Tipo de columna SQL[1]	Tipo de datos C/C++	Descripción del tipo de columna SQL

VARGRAPHIC(*n*)
(464 ó 465)
	struct tag {
	short int;
	wchar_t [*n*]
	}
	
	1<=*n*<=16 336
		Serie de caracteres de doble byte y longitud variable no terminada en nulo con indicador de longitud de serie de 2 bytes

	Como alternativa, utilice char[*n+1*], donde n es suficientemente grande para contener los datos 1<=*n*<=16 336	Serie de caracteres de doble byte y longitud variable terminada en nulo **Nota:** se le asigna un tipo SQL de 400/401.

LONG VARGRAPHIC
(472 ó 473)
	struct tag {
	short int;
	wchar_t [*n*]
	}
	
	16 337<=*n*<=16 350
		Serie de caracteres de doble byte y longitud variable no terminada en nulo con indicador de longitud de serie de 2 bytes

Nota: los tipos de datos siguientes sólo están disponibles en los entornos DBCS o EUC.		
DBCLOB(*n*) (412 ó 413)	sql type is dbclob(*n*) 1<=*n*<=1 073 741 823	Serie de caracteres de doble byte y longitud variable no terminada en nulo con indicador de longitud de serie de 4 bytes
Variable de localizador DBCLOB[6] (968 ó 969)	sql type is dbclob_locator	Identifica las entidades DBCLOB que residen en el servidor
Variable de referencia de archivos DBCLOB[6] (924 ó 925)	sql type is dbclob_file	Descriptor del archivo que contiene datos DBCLOB

Notas:

1. El primer número que se encuentra bajo **Tipo de columna SQL** indica que no se proporciona una variable de indicador, y el segundo número indica que se proporciona una variable de indicador. Se necesita una variable de indicador para indicar los valores nulos (NULL) o para contener la longitud de una serie truncada. Éstos son los valores que aparecerán en el campo SQLTYPE del SQLDA para estos tipos de datos.
2. Por razones de compatibilidad entre plataformas, utilice sqlint32. En las plataformas UNIX de 64 bits, ″long″ es un entero de 64 bits. En los sistemas operativos Windows de 64 bits y en las plataformas UNIX de 32 bits, ″long″ es un entero de 32 bits.
3. Por razones de compatibilidad entre plataformas, utilice sqlint64. El archivo de cabecera sqlsystm.h de DB2 Universal Database definirá el tipo sqlint64 como ″__int64″ en la plataforma Windows NT cuando se utilice el compilador de Microsoft, como ″long long″ en las plataformas UNIX de 32 bits y como ″long″ en las plataformas UNIX de 64 bits.
4. FLOAT(*n*) donde 0 < *n* < 25 es un sinónimo de REAL. La diferencia entre REAL y DOUBLE en el SQLDA es el valor de la longitud (4 ó 8).
5. Los tipos SQL siguientes son sinónimos de DOUBLE:
   - FLOAT
   - FLOAT(*n*) donde 24 < *n* < 54 es
   - DOUBLE PRECISION
6. Éste no es un tipo de columna, sino un tipo de variable del lenguaje principal.

A continuación mostramos normas adicionales para los tipos de datos C/C++ soportados:

- El tipo de datos `char` se puede declarar como `char` o `unsigned char`.
- El gestor de bases de datos procesa los datos de serie de caracteres de longitud variable y terminados en nulo del tipo char[*n*] (tipo de datos 460) como VARCHAR(*m*).
  - Si LANGLEVEL es SAA1, la longitud de la variable del lenguaje principal *m* es igual a la longitud de la serie de caracteres *n* en char[*n*] o al número de bytes que preceden al primer terminador nulo (\0), el valor más pequeño de ambos.
  - Si LANGLEVEL es MIA, la longitud de la variable del lenguaje principal *m* es igual al número de bytes que preceden al primer terminador nulo (\0).
- El gestor de bases de datos procesa el tipo de datos de serie de caracteres de gráficos y longitud variable terminados en nulo, `wchar_t[n]` o `sqldbchar[n]` (tipo de datos 400), como VARGRAPHIC(*m*).
  - Si LANGLEVEL es SAA1, la longitud de la variable del lenguaje principal *m* es igual a la longitud de la serie de caracteres *n* en `wchar_t[n]` o `sqldbchar[n]`, o al número de caracteres que preceden al primer terminador nulo de gráficos, el valor más pequeño de ambos.
  - Si LANGLEVEL es MIA, la longitud de la variable del lenguaje principal *m* es igual al número de caracteres que preceden al primer terminador nulo de gráficos.
- No se soportan los tipos de datos numéricos sin signo.
- El tipo de datos `int` de C/C++ no está permitido, puesto que su representación interna depende de la máquina.

Conceptos relacionados:

- "Sección declare de SQL con variables del lenguaje principal para C y C++" en la página 216

FOR BIT DATA en C y C++

No se debe utilizar el tipo de serie estándar de C o C++, 460, para las columnas designadas como FOR BIT DATA. El gestor de bases de datos trunca este tipo de datos cuando se encuentra un carácter nulo. Utilice las estructuras VARCHAR (tipo de SQL 448) o CLOB (tipo de SQL 408).

Conceptos relacionados:

- "Sección declare de SQL con variables del lenguaje principal para C y C++" en la página 216

Consulta relacionada:

• “Tipos de datos SQL soportados en C y C++” en la página 218

Tipos de datos de C y C++ para procedimientos, funciones y métodos

La tabla siguiente lista las correlaciones soportadas entre tipos de datos SQL y tipos de datos C para procedimientos, UDF y métodos.

Tabla 14. Tipos de datos SQL correlacionados con declaraciones C/C++

Tipo de columna SQL	Tipo de datos C/C++	Descripción del tipo de columna SQL
SMALLINT (500 ó 501)	short	Entero con signo de 16 bits
INTEGER (496 ó 497)	sqlint32	Entero con signo de 32 bits
BIGINT (492 ó 493)	sqlint64	Entero con signo de 64 bits
REAL (480 ó 481)	float	Coma flotante de precisión simple
DOUBLE (480 ó 481)	double	Coma flotante de precisión doble
DECIMAL(*p*,*s*) (484 ó 485)	No soportado.	Para pasar un valor decimal, defina el parámetro como de tipo de datos moldeable DECIMAL (por ejemplo, CHAR o DOUBLE) y moldee explícitamente el argumento para este tipo.
CHAR(*n*) (452 ó 453)	char[*n+1*] donde n es suficientemente grande para contener los datos 1<=*n*<=254	Serie de caracteres de longitud fija terminada en nulo
CHAR(*n*) FOR BIT DATA (452 ó 453)	char[*n+1*] donde n es suficientemente grande para contener los datos 1<=*n*<=254	Serie de caracteres de longitud fija
VARCHAR(*n*) (448 ó 449) (460 ó 461)	char[*n+1*] donde n es suficientemente grande para contener los datos 1<=*n*<=32 672	Serie de longitud variable terminada en nulo
VARCHAR(*n*) FOR BIT DATA (448 ó 449)	struct { sqluint16 length; char[*n*] } 1<=*n*<=32 672	Serie de caracteres de longitud variable no terminada en nulo
LONG VARCHAR (456 ó 457)	struct { sqluint16 length; char[*n*] } 32 673<=*n*<=32 700	Serie de caracteres de longitud variable no terminada en nulo

Tabla 14. Tipos de datos SQL correlacionados con declaraciones C/C++ (continuación)

Tipo de columna SQL	Tipo de datos C/C++	Descripción del tipo de columna SQL

CLOB(*n*)
(408 ó 409)
	struct {
	sqluint32 length;
	char data[n];
	}
	
	1<=*n*<=2 147 483 647
		Serie de caracteres de longitud variable no terminada en nulo con indicador de longitud de serie de 4 bytes

BLOB(*n*)
(404 ó 405)
	struct {
	sqluint32 length;
	char data[n];
	}
	
	1<=*n*<=2 147 483 647
		Serie binaria de longitud variable no terminada en nulo con indicador de longitud de serie de 4 bytes

DATE (384 ó 385)	char[11]	Formato de caracteres terminado en nulo
TIME (388 ó 389)	char[9]	Formato de caracteres terminado en nulo
TIMESTAMP (392 ó 393)	char[27]	Formato de caracteres terminado en nulo
Nota: los tipos de datos siguientes sólo están disponibles en los entornos DBCS o EUC si se compilan previamente con la opción WCHARTYPE NOCONVERT.		
GRAPHIC(*n*) (468 ó 469)	sqldbchar[*n+1*] donde n es suficientemente grande para contener los datos 1<=*n*<=127	Serie de caracteres de doble byte y longitud fija terminada en nulo
VARGRAPHIC(*n*) (400 ó 401)	sqldbchar[*n+1*] donde n es suficientemente grande para contener los datos 1<=*n*<=16 336	Serie de caracteres de doble byte y longitud variable no terminada en nulo

LONG VARGRAPHIC
(472 ó 473)
	struct {
	sqluint16 length;
	sqldbchar[*n*]
	}
	
	16 337<=*n*<=16 350
		Serie de caracteres de doble byte y longitud variable no terminada en nulo

DBCLOB(*n*)
(412 ó 413)
	struct {
	sqluint32 length;
	sqldbchar data[n];
	}
	
	1<=*n*<=1 073 741 823
		Serie de caracteres de longitud variable no terminada en nulo con indicador de longitud de serie de 4 bytes

Variables SQLSTATE y SQLCODE en C y C++

Cuando se utiliza la opción de precompilación LANGLEVEL con un valor de SQL92E, se pueden incluir las dos declaraciones siguientes como variables del lenguaje principal:

```
EXEC SQL BEGIN DECLARE SECTION;
  char      SQLSTATE[6]
  sqlint32  SQLCODE;

:
EXEC SQL END DECLARE SECTION;
```

Si no se especifica ninguna de ellas, se supone la declaración SQLCODE durante el paso de precompilación. Observe que, si se utiliza esta opción, no se debe especificar la sentencia INCLUDE SQLCA.

En una aplicación formada por varios archivos fuente, las variables SQLCODE y SQLSTATE se pueden definir en el primer archivo fuente, tal como en el ejemplo anterior. Los archivos fuente posteriores deben modificar las definiciones del modo siguiente:

```
extern sqlint32 SQLCODE;
extern char     SQLSTATE[6];
```

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

Objetivo del acceso a base de datos de varias hebras

Una característica de algunos sistemas operativos es la posibilidad de ejecutar varias hebras de ejecución en un solo proceso. Las diversas hebras permiten que una aplicación maneje sucesos asíncronos y facilitan la creación de aplicaciones dirigidas por sucesos sin recurrir a esquemas de tramas. La siguiente información explica cómo trabaja el gestor de bases de datos con varias hebras y se relacionan algunas directrices de diseño que conviene recordar.

Si no está familiarizado con los términos relacionados con el desarrollo de aplicaciones de varias hebras (como sección crítica y semáforo), consulte la documentación sobre programación correspondiente al sistema operativo.

Una aplicación DB2 puede ejecutar sentencias de SQL para varias hebras utilizando *contextos*. Un contexto es el entorno desde el que una aplicación ejecuta todas las sentencias de SQL y las llamadas a las API. Todas las conexiones, unidades de trabajo y otros recursos de base de datos están asociados a un contexto específico. Cada contexto está asociado a una o más hebras de una aplicación.

Para cada sentencia de SQL ejecutable en un contexto, la primera llamada a los servicios de tiempo de ejecución siempre intenta obtener un enganche. Si esta operación resulta satisfactoria, prosigue el proceso. Si no lo es (porque una sentencia de SQL de otra hebra del mismo contexto ya tiene el enganche), se bloquea la llamada en un semáforo de señalización hasta que entra en funciones dicho semáforo, en cuyo momento la llamada obtiene el enganche y prosigue el proceso. Se mantiene en enganche hasta que se ha completado el proceso, momento en que lo libera la última llamada a los servicios de tiempo de ejecución que se ha generado para esa sentencia de SQL en particular.

El resultado neto es que cada sentencia de SQL de un contexto se ejecuta como una unidad atómica, aunque puede que haya otras hebras que estén

intentando ejecutar sentencias de SQL al mismo tiempo. Esta acción asegura que las distintas hebras que puedan actuar a la vez no alteran las estructuras internas de los datos. Las API también utilizan el enganche utilizado por los servicios de tiempo de ejecución; por consiguiente, las API tienen las mismas restricciones que las rutinas de los servicios de tiempo de ejecución dentro de cada contexto.

Por omisión, todas las aplicaciones tienen un solo contexto que se utiliza para todos los accesos a bases de datos. Aunque que esto resulta perfecto para una aplicación de una única hebra, la colocación en serie de las sentencias de SQL hace que un solo contexto resulte inadecuado para una aplicación de varias hebras. Utilizando las siguientes API de DB2, la aplicación puede conectar un contexto separado a cada hebra y permitir que se pasen contextos entre hebras:

- `sqleSetTypeCtx()`
- `sqleBeginCtx()`
- `sqleEndCtx()`
- `sqleAttachToCtx()`
- `sqleDetachFromCtx()`
- `sqleGetCurrentCtx()`
- `sqleInterruptCtx()`

En un proceso, se pueden intercambiar contextos entre hebras, pero no se pueden intercambiar entre procesos. Un uso de varios contextos consiste en proporcionar soporte de transacciones simultáneas.

Conceptos relacionados:

- “Transacciones simultáneas” en la página 469

Consulta relacionada:

- “sqleAttachToCtx - Attach to Context” en el manual *Administrative API Reference*
- “sqleBeginCtx - Create and Attach to an Application Context” en el manual *Administrative API Reference*
- “sqleDetachFromCtx - Detach From Context” en el manual *Administrative API Reference*
- “sqleEndCtx - Detach and Destroy Application Context” en el manual *Administrative API Reference*
- “sqleGetCurrentCtx - Get Current Context” en el manual *Administrative API Reference*
- “sqleInterruptCtx - Interrupt Context” en el manual *Administrative API Reference*
- “sqleSetTypeCtx - Set Application Context Type” en el manual *Administrative API Reference*

Ejemplos relacionados:

- "dbthrds.sqc -- How to use multiple context APIs on UNIX (C)"
- "dbthrds.sqC -- How to use multiple context APIs on UNIX (C++)"

Recomendaciones para utilizar varias hebras

Cuando acceda a una base de datos desde aplicaciones de varias hebras, siga estas directrices:

- Coloque en serie la alteración de estructuras de datos.

  Las aplicaciones se deben asegurar de que las estructuras de datos, definidas por el usuario y utilizadas por las sentencias de SQL y por las rutinas del gestor de bases de datos, no se vean alteradas por una hebra mientras se esté procesando una sentencia de SQL o una rutina del gestor de bases de datos en otra hebra. Por ejemplo, no permita que una hebra reasigne una SQLDA mientras la está utilizando una sentencia de SQL en otra hebra.

- Considere la posibilidad de utilizar estructuras de datos separadas.

  Puede resultar más fácil asignar a cada hebra sus propias estructuras de datos definidas por el usuario, a fin de evitar que se coloque en serie su uso. Esta directriz es especialmente cierta para la SQLCA, la cual utilizan no sólo todas las sentencias de SQL ejecutables, sino también todas las rutinas del gestor de bases de datos. Existen tres alternativas para evitar este problema con la SQLCA:

  – Utilice EXEC SQL INCLUDE SQLCA, pero añada `struct sqlca sqlca` al comienzo de cualquier rutina que utilice cualquier hebra que no sea la primera.

  – Coloque EXEC SQL INCLUDE SQLCA dentro de cada rutina que contenga SQL, en lugar de hacerlo en el ámbito global.

  – Sustituya EXEC SQL INCLUDE SQLCA por `#include "sqlca.h"` y luego añada `"struct sqlca sqlca"` al comienzo de cualquier rutina que utilice SQL.

Consideraciones sobre página de códigos y código de país/región para aplicaciones UNIX de varias hebras

En AIX, Solaris Operating Environment, HP-UX y Silicon Graphics IRIX, se han efectuado cambios en las funciones que se utilizan para ejecutar consultas en tiempo de ejecución de la página de códigos y del código de país/región, a fin de utilizarlas para conectar con una base de datos. Ahora estas funciones están a salvo de las hebras, pero pueden crear alguna contención de bloqueos

(dando como resultado una degradación del rendimiento) en una aplicación de varias hebras que utilice un gran número de conexiones de base de datos simultáneas.

Puede utilizar la variable de entorno DB2_FORCE_NLS_CACHE para eliminar la posibilidad de contención de bloqueos en las aplicaciones de varias hebras. Si DB2_FORCE_NLS_CACHE tiene el valor TRUE, la primera vez que una hebra accede a la información de página de códigos y código de país/región, ésta se guarda. A partir de este punto, cualquier otra hebra que solicite esta información utilizará la que se encuentra en la antememoria. Guardando esta información, se suprime la contención de bloqueos y, en determinadas situaciones, se obtiene una mejora en el rendimiento.

No debe asignar a DB2_FORCE_NLS_CACHE el valor TRUE si la aplicación cambia los valores de entorno nacional entre conexiones. Si se produce esta situación, se devolverá la información del entorno nacional original aunque se hayan cambiado estos valores. En general, las aplicaciones de varias hebras no cambiarán los valores de entorno nacional, lo cual asegura que la aplicación sigue estando a salvo de hebras.

Conceptos relacionados:

- "DB2 registry and environment variables" en el manual *Administration Guide: Performance*

Resolución de problemas de aplicaciones de varias hebras

Las secciones siguientes describen problemas que se pueden producir con una aplicación de varias hebras y cómo evitarlos.

Problemas potenciales con varias hebras

Una aplicación que utiliza varias hebras es, comprensiblemente, más compleja que una aplicación de una sola hebra. Esta complejidad adicional puede conducir potencialmente a algunos problemas inesperados. Cuando escriba una aplicación de varias hebras, tenga precaución con lo siguiente:

- Dependencias de base de datos entre dos o más contextos.

  Cada contexto de una aplicación tiene sus propios recursos de base de datos, incluidos los bloqueos sobre objetos de base de datos. Esta característica posibilita que dos contextos, si están accediendo al mismo objeto de base de datos, lleguen a un punto muerto. El gestor de bases de datos detectará el punto muerto. Uno de los contextos recibirá SQLCODE -911 y se retrotraerá la unidad de trabajo del mismo.

- Dependencias de aplicaciones entre dos o más contextos.

  Tenga cuidado con las técnicas de programación que establecen dependencias entre contextos. Los enganches, los semáforos y las secciones

críticas son ejemplos de técnicas de programación que pueden establecer dichas dependencias. Si una aplicación tiene dos contextos que tienen dependencias, tanto de aplicación como de base de datos, entre los contextos, es posible que la aplicación llegue a un punto muerto. Si algunas de las dependencias están fuera del gestor de bases de datos, no se detecta el punto muerto, por lo que la aplicación se suspende o se cuelga.

Conceptos relacionados:

• "Cómo evitar puntos muertos para varios contextos" en la página 231

Cómo evitar puntos muertos para varios contextos

Puesto que el gestor de bases de datos no detecta puntos muertos entre hebras, diseñe y codifique la aplicación de modo que impida (o al menos evite) puntos muertos.

Como ejemplo de un punto muerto que el gestor de bases de datos no detectaría piense en una aplicación que tiene dos contextos y ambos acceden a una estructura de datos común. Para evitar problemas en que ambos contextos cambien simultáneamente la estructura de datos, la estructura de datos se protege mediante un semáforo. Los contextos tienen el aspecto siguiente:

```
contexto 1
SELECT * FROM TAB1 FOR UPDATE....
UPDATE TAB1 SET....
get semaphore
access data structure
release semaphore
COMMIT

contexto 2
get semaphore
access data structure
SELECT * FROM TAB1...
release semaphore
COMMIT
```

Suponga que el primer contexto ejecuta satisfactoriamente las sentencias SELECT y UPDATE, mientras que el segundo contexto obtiene el semáforo y accede a la estructura de datos. Ahora, el primer contexto intenta obtener el semáforo, pero no puede porque el segundo contexto lo está reteniendo. A continuación, el segundo contexto intenta leer una fila de la tabla TAB1, pero se detiene en un bloqueo de la base de datos mantenido por el primer contexto. La aplicación se encuentra ahora en un estado en que el contexto 1 no puede terminar antes de que lo haga el contexto 2, y el contexto 2 está esperando a que el contexto 1 termine. La aplicación está en un punto muerto

pero, puesto que el gestor de bases de datos no sabe nada de la dependencia del semáforo, no se retrotraerá ninguno de los contextos. La dependencia no resuelta deja la aplicación suspendida.

Puede evitar el punto muerto que se produciría en el ejemplo anterior de varias formas.

- Libere todos los bloqueos mantenidos antes de obtener el semáforo.

  Cambie el código del contexto 1 de forma que realice una confirmación antes de obtener el semáforo.
- No codifique sentencias de SQL en una sección protegida por semáforos.

  Cambie el código del contexto 2 de forma que libere el semáforo antes de ejecutar SELECT.
- Codifique todas las sentencias de SQL dentro de semáforos.

  Cambie el código del contexto 1 de forma que obtenga el semáforo antes de ejecutar la sentencia SELECT. Aunque esta técnica funcionará, no es muy recomendable, puesto que los semáforos colocarán en serie el acceso al gestor de bases de datos, lo cual invalidará potencialmente las ventajas de utilizar varias hebras.
- Establezca el parámetro de configuración de la base de datos *locktimeout* en un valor distinto de -1.

  Aunque un valor distinto de -1 no impedirá el punto muerto, permitirá que se reanude la ejecución. Eventualmente, se retrotraerá el contexto 2, puesto que no puede obtener el bloqueo solicitado. Cuando maneje el error de retrotracción, el contexto 2 debe liberar el semáforo. Una vez que se haya liberado el semáforo, el contexto 1 podrá continuar y el contexto 2 será libre de reintentar su trabajo.

Las técnicas para evitar puntos muertos se describen en términos del ejemplo, pero las puede aplicar a todas las aplicaciones de varias hebras. En general, trate el gestor de bases de datos tal como trataría cualquier recurso protegido y no experimentará problemas con las aplicaciones de varias hebras.

Conceptos relacionados:

- “Problemas potenciales con varias hebras” en la página 230

Capítulo 8. Programación en COBOL

Consideraciones sobre la programación en COBOL

En las secciones siguientes se explican las consideraciones especiales sobre la programación en el lenguaje principal. Se incluye información sobre las restricciones de lenguaje, los archivos include específicos del lenguaje principal, la incorporación de sentencias de SQL, las variables del lenguaje principal y los tipos de datos soportados para las variables del lenguaje principal. Para obtener información sobre cómo incorporar sentencias de SQL, las restricciones del lenguaje y los tipos de datos soportados para las variables del lenguaje principal, consulte la documentación de Micro Focus COBOL.

Consulta relacionada:

- "Programas de ejemplo COBOL" en el manual *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

Restricciones de lenguaje en COBOL

Todos los punteros de API tienen una longitud de 4 bytes. Todas las variables enteras utilizadas como parámetros de valor en las llamadas a las API se deben declarar con una cláusula USAGE COMP-5.

Acceso a bases de datos de varias hebras en COBOL

COBOL no da soporte al acceso a bases de datos de varias hebras.

Archivos de entrada y salida para COBOL

Por omisión, el archivo de entrada tiene la extensión `.sqb`, pero si se utiliza la opción de precompilación TARGET (TARGET ANSI_COBOL, TARGET IBMCOB, TARGET MFCOB o TARGET MFCOB16), el archivo de entrada puede tener la extensión que elija el usuario.

Por omisión, el archivo de salida tiene la extensión `.cbl`, pero se puede utilizar la opción de precompilación OUTPUT para especificar un nombre y una vía de acceso nuevos para el archivo fuente de salida modificado.

Archivos include para COBOL

Los archivos include específicos del lenguaje principal para COBOL tienen la extensión de archivo `.cbl`. Si se utiliza la característica ″Soporte para tipo de datos de sistema principal System/390″ del compilador COBOL de IBM, los archivos include de DB2 para las aplicaciones se encuentran en el directorio siguiente:

```
$HOME/sqllib/include/cobol_i
```

Si crea los programas de ejemplo de DB2 con los archivos de script suministrados, deben cambiar la vía de acceso de archivos include especificada en los archivos de script por el directorio `cobol_i`, y no por el directorio `cobol_a`.

Si **no** utiliza la característica ″Soporte para tipo de datos de sistema principal System/390″ del compilador COBOL de IBM, o si utiliza una versión anterior de este compilador, los archivos include de DB2 para las aplicaciones se encuentran en el directorio siguiente:

```
$HOME/sqllib/include/cobol_a
```

A continuación se describen los archivos include destinados a su uso en las aplicaciones del usuario.

SQL (sql.cbl) Este archivo incluye prototipos específicos del lenguaje para el vinculador, el precompilador y las API de recuperación de mensajes de error. También define constantes del sistema.

SQLAPREP (sqlaprep.cbl)
Este archivo contiene definiciones necesarias para escribir su propio precompilador.

SQLCA (sqlca.cbl)
Este archivo define la estructura del Área de comunicaciones de SQL (SQLCA). El SQLCA contiene variables que utiliza el gestor de bases de datos para proporcionar a una aplicación información de error sobre la ejecución de sentencias de SQL y llamadas a API.

SQLCA_92 (sqlca_92.cbl)
Este archivo contiene una versión que se ajusta a FIPS SQL92 Entry Level de la estructura Área de comunicaciones de SQL (SQL Communications Area (SQLCA)). Debe incluir este archivo en lugar del archivo `sqlca.cbl` cuando escriba aplicaciones DB2 que se ajusten al estándar FIPS SQL92 Entry Level. El precompilador de DB2 incluye automáticamente el archivo `sqlca_92.cbl` cuando la opción LANGLEVEL del precompilador se establece en SQL92E.

SQLCODES (sqlcodes.cbl)
Este archivo define constantes para el campo SQLCODE de la estructura SQLCA.

SQLDA (sqlda.cbl)
Este archivo define la estructura del Área de descriptor de SQL (SQLDA). El SQLDA se utiliza para pasar datos entre una aplicación y el gestor de bases de datos.

SQLEAU (sqleau.cbl)
Este archivo contiene definiciones de constantes y de estructuras necesarias para las API de auditoría de seguridad de DB2. Si utiliza estas API, tiene que incluir este archivo en el programa. Este archivo también contiene definiciones de constantes y de valores de palabras clave para los campos del registro de seguimiento de auditoría. Programas externos o de extracción de seguimiento de auditoría de proveedores pueden utilizar estas definiciones.

SQLENV (sqlenv.cbl)
Este archivo define llamadas específicas del lenguaje para las API del entorno de bases de datos y las estructuras, constantes y códigos de retorno correspondientes a dichas interfaces.

SQLETSD (sqletsd.cbl)

Este archivo define la estructura Descriptor de espacio de tablas (Table Space Descriptor), SQLETSDESC, que se pasa a la API para Crear base de datos, sqlgcrea.

SQLE819A (sqle819a.cbl)

Si la página de códigos de la base de datos es 819 (ISO Latin-1), esta secuencia clasifica series de caracteres que no son FOR BIT DATA según la clasificación binaria CCSID 500 (EBCDIC internacional) del sistema principal. La API CREATE DATABASE utiliza este archivo.

SQLE819B (sqle819b.cbl)

Si la página de códigos de la base de datos es 819 (ISO Latin-1), esta secuencia clasifica series de caracteres que no son FOR BIT DATA según la clasificación binaria CCSID 037 (EBCDIC inglés de EE.UU.) del sistema principal. La API CREATE DATABASE utiliza este archivo.

SQLE850A (sqle850a.cbl)

Si la página de códigos de la base de datos es 850 (ASCII Latin-1), esta secuencia clasifica series de caracteres que no son FOR BIT DATA según la clasificación binaria CCSID 500 (EBCDIC internacional) del sistema principal. La API CREATE DATABASE utiliza este archivo.

SQLE850B (sqle850b.cbl)

Si la página de códigos de la base de datos es 850 (ASCII Latin-1), esta secuencia clasifica series de caracteres que no son FOR BIT DATA según la clasificación binaria CCSID 037 (EBCDIC inglés de EE.UU.) del sistema principal. La API CREATE DATABASE utiliza este archivo.

SQLE932A (sqle932a.cbl)

Si la página de códigos de la base de datos es 932 (ASCII japonés), esta secuencia clasifica series de caracteres que no son FOR BIT DATA según la clasificación binaria CCSID 5035 (EBCDIC japonés) del sistema principal. La API CREATE DATABASE utiliza este archivo.

SQLE932B (sqle932b.cbl)

Si la página de códigos de la base de datos es 932 (ASCII japonés), esta secuencia clasifica series de caracteres que no son FOR BIT DATA según la clasificación binaria CCSID 5026 (EBCDIC japonés) del sistema principal. La API CREATE DATABASE utiliza este archivo.

SQL1252A (sql1252a.cbl)

Si la página de códigos de la base de datos es 1252 (Windows

Latin-1), esta secuencia clasifica series de caracteres que no son FOR BIT DATA según la clasificación binaria CCSID 500 (EBCDIC internacional) del sistema principal. La API CREATE DATABASE utiliza este archivo.

SQL1252B (sql1252b.cbl)
Si la página de códigos de la base de datos es 1252 (Windows Latin-1), esta secuencia clasifica series de caracteres que no son FOR BIT DATA según la clasificación binaria CCSID 037 (EBCDIC inglés de EE.UU.) del sistema principal. La API CREATE DATABASE utiliza este archivo.

SQLMON (sqlmon.cbl)
Este archivo define llamadas específicas del lenguaje para las API del supervisor del sistema de bases de datos y las estructuras, constantes y códigos de retorno correspondientes a dichas interfaces.

SQLMONCT (sqlmonct.cbl)
Este archivo contiene definiciones de constantes y definiciones de estructuras de datos locales necesarias para llamar a las API del Supervisor del sistema de bases de datos.

SQLSTATE (sqlstate.cbl)
Este archivo define constantes correspondientes al campo SQLSTATE de la estructura SQLCA.

SQLUTBCQ (sqlutbcq.cbl)
Este archivo define la estructura de datos (Consulta de contenedor de espacio de tablas (Table Space Container Query), SQLB-TBSCONTQRY-DATA, que se utiliza con las API de consulta del contenedor del espacio de tablas, sqlgstsc, sqlgftcq y sqlgtcq.

SQLUTBSQ (sqlutbsq.cbl)
Este archivo define la estructura de datos Consulta de espacio de tablas (Table Space Query), SQLB-TBSTQRY-DATA, que se utiliza con las API de consulta del espacio de tablas, sqlgstsq, sqlgftsq y sqlgtsq.

SQLUTIL (sqlutil.cbl)
Este archivo define las llamadas específicas del lenguaje correspondientes a las API de programas de utilidad y las estructuras, constantes y códigos necesarios para dichas interfaces.

Sentencias de SQL incorporado en COBOL

Las sentencias de SQL incorporado constan de los tres elementos siguientes:

Elemento	Sintaxis correcta en COBOL
Par de palabras clave	EXEC SQL
Serie de la sentencia	Cualquier sentencia de SQL válida
Terminador de la sentencia	END-EXEC.

Por ejemplo:

```
EXEC SQL SELECT col INTO :hostvar FROM table END-EXEC.
```

Se aplican las normas siguientes a las sentencias de SQL incorporado:

- Las sentencias de SQL ejecutables se deben colocar en PROCEDURE DIVISION. Las sentencias de SQL pueden ir precedidas de un nombre de párrafo, al igual que una sentencia de COBOL.
- Las sentencias de SQL puede empezar en el Área A (columnas 8 a 11) o en el Área B (columnas 12 a 72).
- Inicie cada sentencia de SQL con EXEC SQL y termínela con END-EXEC. El precompilador SQL incluye cada una de las sentencias de SQL como comentarios en el archivo fuente modificado.
- Debe utilizar el terminador de sentencias de SQL. De no utilizarlo, el precompilador seguirá hasta el siguiente terminador de la aplicación. Esto puede producir errores indeterminados.
- Están permitidos los comentarios de SQL en cualquier línea que forme parte de una sentencia de SQL incorporado. Estos comentarios no están permitidos en las sentencias que se ejecutan de forma dinámica. El formato de un comentario de SQL consiste en un guión doble (--), seguido de una serie compuesta por cero o más caracteres y terminada por un fin de línea. No coloque comentarios de SQL detrás del terminador de la sentencia de SQL, puesto que ocasionarían errores de compilación debido a que parecería que forman parte del lenguaje COBOL.
- Los comentarios COBOL están permitidos *casi* en cualquier lugar de una sentencia de SQL incorporado. Las excepciones son:
  - No se permiten comentarios entre EXEC y SQL.
  - No se permiten comentarios en las sentencias que se ejecutan dinámicamente.
- Las sentencias de SQL siguen las mismas normas de continuación de línea que el lenguaje COBOL. No obstante, no divida el par de palabras clave EXEC SQL en distintas líneas.
- No utilice la sentencia COPY de COBOL para incluir archivos que contengan sentencias de SQL. Las sentencias de SQL se precompilan antes de que se compile el módulo. El precompilador pasará por alto la sentencia COPY de COBOL. En su lugar, utilice la sentencia INCLUDE de SQL para incluir dichos archivos.

Para localizar el archivo INCLUDE, el precompilador COBOL de DB2 busca en primer lugar en el directorio actual y, a continuación, en los directorios especificados por la variable de entorno DB2INCLUDE. Considere los ejemplos siguientes:

– `EXEC SQL INCLUDE payroll END-EXEC.`

  Si el archivo especificado en la sentencia INCLUDE no está encerrado entre comillas, como en el ejemplo anterior, el precompilador C busca `payroll.sqb`, luego `payroll.cpy` y luego `payroll.cbl` en cada uno de los directorios que mira.

– `EXEC SQL INCLUDE 'pay/payroll.cbl' END-EXEC.`

  Si el nombre de archivo está encerrado entre comillas, como en el caso anterior, no se añade ninguna extensión al nombre.

  Si el nombre del archivo entrecomillado no contiene una vía de acceso absoluta, se utiliza el contenido de DB2INCLUDE para buscar el archivo, añadiéndole como prefijo la vía de acceso especificada en el nombre del archivo INCLUDE. Por ejemplo, con DB2 para AIX, si DB2INCLUDE se establece en '`/disk2:myfiles/cobol`', el precompilador busca '`./pay/payroll.cbl`', luego '`/disk2/pay/payroll.cbl`' y finalmente '`./myfiles/cobol/pay/payroll.cbl`'. En los mensajes del precompilador se muestra la vía de acceso en la que se encuentra realmente el archivo. En las plataformas Windows, sustituya las barras inclinadas invertidas (\) del ejemplo anterior por barras inclinadas.

Nota: el procesador de línea de mandatos de DB2 coloca en antememoria el valor de DB2INCLUDE. Para cambiar el valor de DB2INCLUDE después de haber emitido mandatos del CLP, entre el mandato TERMINATE, luego vuelva a conectar con la base de datos y realice una precompilación del modo habitual.

- Para continuar una constante de serie en la línea siguiente, en la columna 7 de la línea de continuación debe aparecer un '-' y en la columna 12 o posteriores debe aparecer un delimitador de serie.
- Los operadores aritméticos de SQL se deben delimitar mediante espacios en blanco.
- Pueden aparecer comentarios de COBOL de línea completa en cualquier lugar del programa, incluso dentro de sentencias de SQL.
- Cuando haga referencia a variables del lenguaje principal en una sentencia de SQL, utilícelas exactamente tal como se han declarado.
- La sustitución de caracteres de espacio en blanco, como caracteres de fin de línea y de tabulación, se produce del siguiente modo:
  – Cuando aparecen fuera de las comillas (pero dentro de sentencias de SQL), los caracteres de fin de línea y tabuladores se sustituyen por un solo espacio.

– Si aparecen dentro de las comillas, los caracteres de fin de línea desaparecen, siempre que se continúe correctamente la serie para un programa COBOL. Los tabuladores no se modifican.

Observe que los caracteres reales utilizados como fin de línea y tabulador varían en función de la plataforma. Por ejemplo, las plataformas basadas en Windows utilizan Retorno de carro/Salto de línea como fin de línea, mientras que los sistemas basados en UNIX sólo utilizan un Salto de línea.

Consulta relacionada:

- Apéndice A, “Sentencias de SQL soportadas” en la página 521

Variables del lenguaje principal en COBOL

Las secciones siguientes describen cómo declarar y utilizar variables del lenguaje principal en programas COBOL.

Variables del lenguaje principal en COBOL

Las variables del lenguaje principal son variables del lenguaje COBOL a las que se hace referencia en las sentencias de SQL. Permiten que una aplicación pase datos de entrada al gestor de bases de datos y reciba datos de salida de éste. Una vez que se ha precompilado la aplicación, el compilador utiliza las variables del lenguaje principal como cualquier otra variable de COBOL.

Conceptos relacionados:

- “Nombres de variables del lenguaje principal en COBOL” en la página 241
- “Declaraciones de variables del lenguaje principal en COBOL” en la página 242

Consulta relacionada:

- “Sintaxis de las variables numéricas del lenguaje principal en COBOL” en la página 242
- “Sintaxis de las variables del lenguaje principal de tipo carácter de longitud fija en COBOL” en la página 243
- “Sintaxis de las variables gráficas del lenguaje principal de longitud fija en COBOL” en la página 245
- “Sintaxis de las variables del lenguaje principal LOB en COBOL” en la página 246
- “Sintaxis de las variables del lenguaje principal de localizador de LOB en COBOL” en la página 248
- “Sintaxis de las variables del lenguaje principal de referencia de archivos en COBOL” en la página 248

Nombres de variables del lenguaje principal en COBOL

El precompilador SQL identifica las variables del lenguaje principal por el nombre declarado para éstas. Se aplican las normas siguientes:

- Especifique nombres de variables con una longitud máxima de 255 caracteres.
- Empiece los nombres de variables con prefijos que no sean SQL, sql, DB2 ni db2, que están reservados para uso del sistema.
- Los elementos FILLER que utilizan las sintaxis de declaración descritas a continuación están permitidos en las declaraciones de variables del lenguaje principal de grupos y el precompilador los pasará por alto. Sin embargo, si se utiliza FILLER más de una vez dentro de una sección DECLARE de SQL, el precompilador fallará. No se pueden incluir elementos FILLER en declaraciones VARCHAR, LONG VARCHAR, VARGRAPHIC ni LONG VARGRAPHIC.
- Puede utilizar guiones en los nombres de variables del lenguaje principal.

  SQL interpreta un guión encerrado entre espacios como un operador de resta. Utilice guiones sin espacios en los nombres de variables del lenguaje principal.
- La cláusula REDEFINES está permitida en las declaraciones de variables del lenguaje principal.
- Las declaraciones de nivel 88 están permitidas en la sección de declaración de variables del lenguaje principal, pero se pasan por alto.

Conceptos relacionados:

- “Declaraciones de variables del lenguaje principal en COBOL” en la página 242

Consulta relacionada:

- “Sintaxis de las variables numéricas del lenguaje principal en COBOL” en la página 242
- “Sintaxis de las variables del lenguaje principal de tipo carácter de longitud fija en COBOL” en la página 243
- “Sintaxis de las variables gráficas del lenguaje principal de longitud fija en COBOL” en la página 245
- “Sintaxis de las variables del lenguaje principal LOB en COBOL” en la página 246
- “Sintaxis de las variables del lenguaje principal de localizador de LOB en COBOL” en la página 248
- “Sintaxis de las variables del lenguaje principal de referencia de archivos en COBOL” en la página 248

Declaraciones de variables del lenguaje principal en COBOL

Se debe utilizar una sección de declaración de SQL para identificar las declaraciones de variables del lenguaje principal. Esta sección alerta al precompilador acerca de las variables del lenguaje principal a las que se puede hacer referencia en sentencias de SQL posteriores.

El precompilador COBOL sólo reconoce un subconjunto de las declaraciones válidas de COBOL.

Tareas relacionadas:

- "Declaración de variables del lenguaje principal de tipo estructurado" en el manual *Guía de desarrollo de aplicaciones: Programación de aplicaciones de servidor*

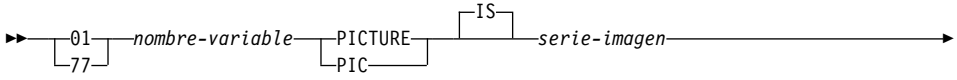
Consulta relacionada:

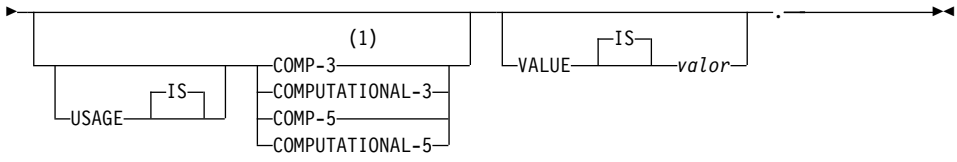
- "Sintaxis de las variables numéricas del lenguaje principal en COBOL" en la página 242
- "Sintaxis de las variables del lenguaje principal de tipo carácter de longitud fija en COBOL" en la página 243
- "Sintaxis de las variables gráficas del lenguaje principal de longitud fija en COBOL" en la página 245
- "Sintaxis de las variables del lenguaje principal LOB en COBOL" en la página 246
- "Sintaxis de las variables del lenguaje principal de localizador de LOB en COBOL" en la página 248
- "Sintaxis de las variables del lenguaje principal de referencia de archivos en COBOL" en la página 248

Sintaxis de las variables numéricas del lenguaje principal en COBOL

A continuación se muestra la sintaxis de las variables numéricas del lenguaje principal.

Sintaxis de las variables numéricas del lenguaje principal en COBOL

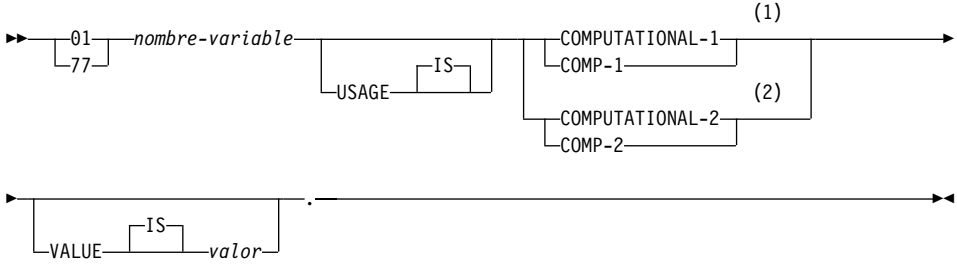




Notas:

1 Una alternativa de COMP-3 es PACKED-DECIMAL.

Coma flotante



Notas:

1 REAL (SQLTYPE 480), Longitud 4

2 DOUBLE (SQLTYPE 480), Longitud 8

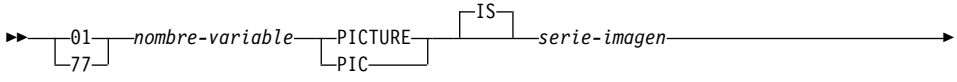
Consideraciones sobre las variables numéricas del lenguaje principal:

1. *Serie-imagen* debe tener uno de los formatos siguientes:
   - S9(m)V9(n)
   - S9(m)V
   - S9(m)
2. Los nueves se pueden expandir (por ejemplo, ″S999″ en lugar de S9(3)″)
3. *m* y *n* deben ser enteros positivos.

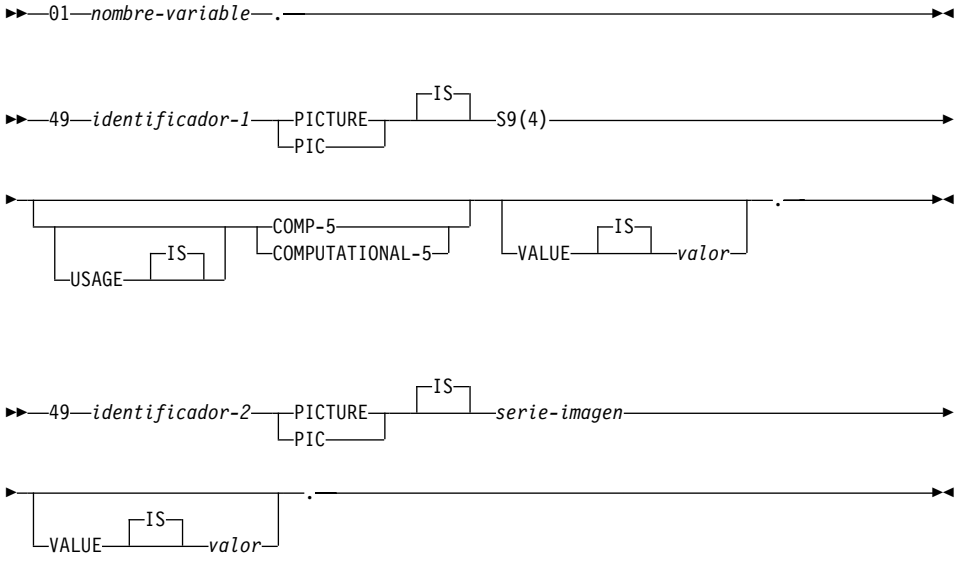
Sintaxis de las variables del lenguaje principal de tipo carácter de longitud fija en COBOL

A continuación se muestra la sintaxis de las variables de tipo carácter del lenguaje principal.

Sintaxis de las variables del lenguaje principal de tipo carácter en COBOL: Longitud fija



Longitud variable



Consideraciones sobre las variables del lenguaje principal de tipo carácter:

1. *Serie-imagen* debe tener el formato *X(m)*. Como alternativa, se pueden expandir las X (por ejemplo, ″XXX″ en lugar de ″X(3)″).
2. *m* va de 1 a 254 para las series de longitud fija.
3. *m* va de 1 a 32 700 para las series de longitud variable.
4. Si *m* es mayor que 32 672, la variable del lenguaje principal se tratará como una serie LONG VARCHAR y su uso puede estar restringido.
5. Utilice X y 9 como caracteres de imagen en cualquier cláusula PICTURE. No están permitidos otros caracteres.
6. Las series de longitud variable constan de un elemento de longitud y un elemento de valor. Puede utilizar nombres aceptables de COBOL para el elemento de longitud y para el elemento de serie. No obstante, haga referencia a las series de longitud variable por el nombres colectivo en las sentencias de SQL.

7. En una sentencia CONNECT, como por ejemplo la que se muestra a continuación, a las variables del lenguaje principal de serie de caracteres en COBOL dbname y userid se les eliminarán los blancos de cola antes del proceso:

```
EXEC SQL CONNECT TO :dbname USER :userid USING :p-word
END-EXEC.
```

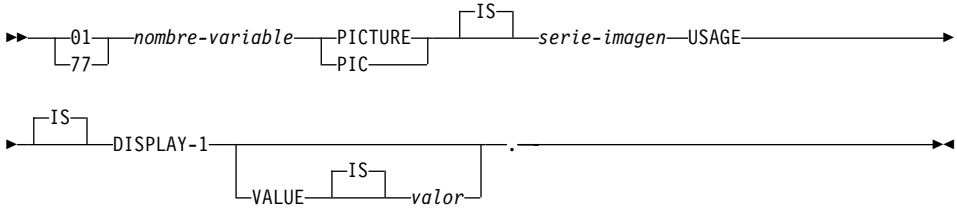
Sin embargo, y puesto que los espacios en blanco pueden ser significativos en las contraseñas, la variable del lenguaje principal p-word se debe declarar como un elemento de datos VARCHAR, de forma que la aplicación pueda indicar explícitamente la longitud significativa de la contraseña para la sentencia CONNECT, del modo siguiente:

```
  EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 dbname PIC X(8).
01 userid PIC X(8).
01 p-word.
    49 L PIC S9(4) COMP-5.
    49 D PIC X(18).
EXEC SQL END DECLARE SECTION END-EXEC.
PROCEDURE DIVISION.
    MOVE "sample" TO dbname.
    MOVE "userid" TO userid.
    MOVE "password" TO D OF p-word.
    MOVE 8          TO L of p-word.
EXEC SQL CONNECT TO :dbname USER :userid USING :p-word
END-EXEC.
```

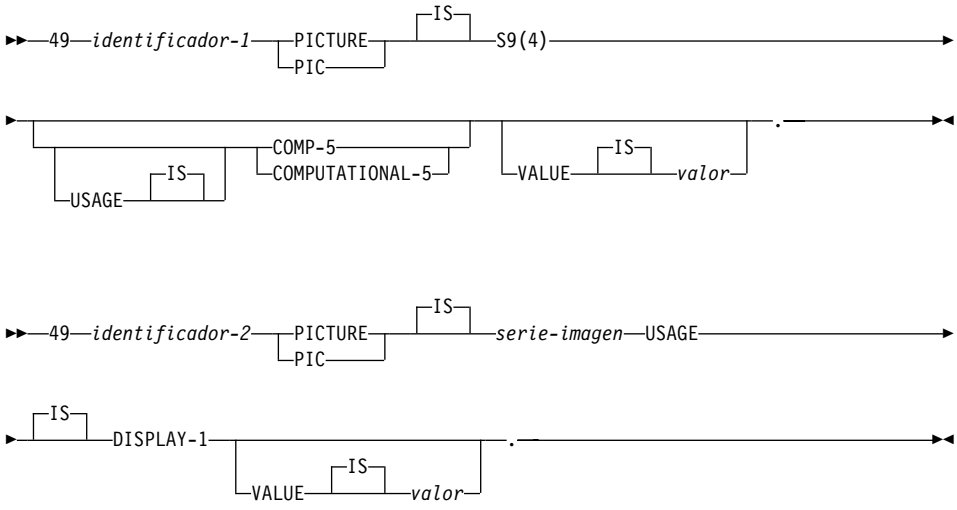
Sintaxis de las variables gráficas del lenguaje principal de longitud fija en COBOL

A continuación se muestra la sintaxis de las variables gráficas del lenguaje principal.

Sintaxis de las variables gráficas del lenguaje principal en COBOL: Longitud fija



Longitud variable



Consideraciones sobre las variables gráficas del lenguaje principal:

1. *Serie-imagen* debe tener el formato *G(m)*. Como alternativa, se pueden expandir las G (por ejemplo, ″GGG″ en lugar de ″G(3)″).
2. *m* va de 1 a 127 para las series de longitud fija.
3. *m* va de 1 a 16 350 para las series de longitud variable.
4. Si *m* es mayor que 16 336, la variable del lenguaje principal se tratará como una serie LONG VARGRAPHIC y su uso puede estar restringido.

Variables de indicador en COBOL

Las variables de indicador se deben declarar con tipo de datos `PIC S9(4) COMP-5`.

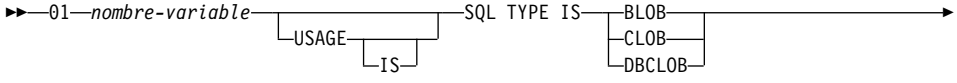
Conceptos relacionados:

- “Tablas de indicadores en COBOL” en la página 251

Sintaxis de las variables del lenguaje principal LOB en COBOL

A continuación se muestra la sintaxis para declarar variables del lenguaje principal de objeto grande (LOB) en COBOL.

Sintaxis de las variables del lenguaje principal LOB en COBOL



Consideraciones sobre las variables del lenguaje principal LOB:

1. Para BLOB y CLOB 1 <= longitud-lob <= 2 147 483 647.
2. Para DBCLOB 1 <= longitud-lob <= 1 073 741 823.
3. SQL TYPE IS, BLOB, CLOB, DBCLOB, K, M, G se pueden especificar en mayúsculas, en minúsculas o en una mezcla de ambas.
4. No se permite la inicialización dentro de la declaración LOB.
5. El nombre de la variable del lenguaje principal es el prefijo de LENGTH y DATA en el código generado por el precompilador.

Ejemplo de BLOB:

La declaración:

```
01 MY-BLOB USAGE IS SQL TYPE IS BLOB(2M).
```

Tiene como resultado la generación de la estructura siguiente:

```
01 MY-BLOB.
49 MY-BLOB-LENGTH PIC S9(9) COMP-5.
49 MY-BLOB-DATA PIC X(2097152).
```

Ejemplo de CLOB:

La declaración:

```
01 MY-CLOB USAGE IS SQL TYPE IS CLOB(125M).
```

Tiene como resultado la generación de la estructura siguiente:

```
01 MY-CLOB.
49 MY-CLOB-LENGTH PIC S9(9) COMP-5.
49 MY-CLOB-DATA PIC X(131072000).
```

Ejemplo de DBCLOB:

La declaración:

```
01 MY-DBCLOB USAGE IS SQL TYPE IS DBCLOB(30000).
```

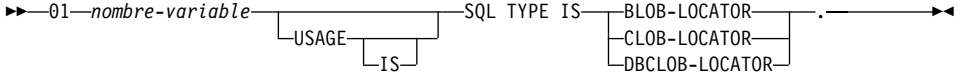
Tiene como resultado la generación de la estructura siguiente:

```
01 MY-DBCLOB.
49 MY-DBCLOB-LENGTH PIC S9(9) COMP-5.
49 MY-DBCLOB-DATA PIC G(30000) DISPLAY-1.
```

Sintaxis de las variables del lenguaje principal de localizador de LOB en COBOL

A continuación se muestra la sintaxis para declarar variables del lenguaje principal de localizador de objeto grande (LOB) en COBOL.

Sintaxis de las variables del lenguaje principal de localizador de LOB en COBOL



Consideraciones sobre las variables del lenguaje principal de localizador de LOB:

1. SQL TYPE IS, BLOB-LOCATOR, CLOB-LOCATOR, DBCLOB-LOCATOR se pueden especificar en mayúsculas, en minúsculas o en una mezcla de ambas.
2. No se permite la inicialización de localizadores.

Ejemplo de localizador de BLOB (existen otros tipos de localizadores de LOB parecidos):

La declaración:

```
01 MY-LOCATOR USAGE SQL TYPE IS BLOB-LOCATOR.
```

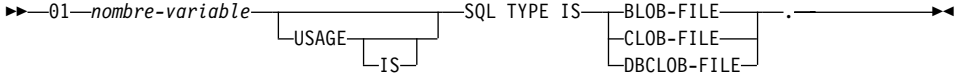
Tiene como resultado la generación de la declaración siguiente:

```
01 MY-LOCATOR PIC S9(9) COMP-5.
```

Sintaxis de las variables del lenguaje principal de referencia de archivos en COBOL

A continuación se muestra la sintaxis para declarar variables del lenguaje principal de referencia de archivos en COBOL.

Sintaxis de las variables del lenguaje principal de referencia de archivos en COBOL



- SQL TYPE IS, BLOB-FILE, CLOB-FILE, DBCLOB-FILE se pueden especificar en mayúsculas, en minúsculas o en una mezcla de ambas.

Ejemplo de referencia de archivos BLOB (existen otros tipos de LOB parecidos):

La declaración:

```
01 MY-FILE USAGE IS SQL TYPE IS BLOB-FILE.
```

Tiene como resultado la generación de la declaración siguiente:

```
01 MY-FILE.
   49 MY-FILE-NAME-LENGTH PIC S9(9) COMP-5.
   49 MY-FILE-DATA-LENGTH PIC S9(9) COMP-5.
   49 MY-FILE-FILE-OPTIONS PIC S9(9) COMP-5.
   49 MY-FILE-NAME PIC X(255).
```

Soporte de estructura del lenguaje principal en COBOL

El precompilador COBOL soporta declaraciones de elementos de datos de grupos en la sección de declaración de variables del lenguaje principal. Entre otras cosas, esto proporciona un sistema taquigráfico para hacer referencia a un conjunto de elementos de datos elementales en una sentencia de SQL. Por ejemplo, se puede utilizar el siguiente elemento de datos de grupo para acceder a algunas de las columnas de la tabla STAFF de la base de datos SAMPLE:

```
01 staff-record.
    05 staff-id       pic s9(4) comp-5.
    05 staff-name.
        49 l          pic s9(4) comp-5.
        49 d          pic x(9).
    05 staff-info.
        10 staff-dept pic s9(4) comp-5.
        10 staff-job  pic x(5).
```

Los elementos de datos de grupos de la sección de declaración pueden tener, como elementos de datos subordinados, cualquiera de los tipos válidos de variable del lenguaje principal descritos anteriormente. Esto incluye todos los tipos de datos numéricos y de tipo carácter, así como los tipos de objeto grande. Los elementos de datos de grupos se pueden anidar hasta un máximo de 10 niveles. Observe que debe declarar los tipos de caracteres VARCHAR con los elementos subordinados al nivel 49, tal como en el ejemplo anterior. Si no están al nivel 49, VARCHAR se trata como elemento de datos de grupos con dos subordinados y está sujeto a las normas de declaración y utilización de elementos de datos de grupos. En el ejemplo anterior, `staff-info` es un elemento de datos de grupos, mientras que `staff-name` es VARCHAR. Se aplica el mismo principio a LONG VARCHAR, VARGRAPHIC y LONG VARGRAPHIC. Puede declarar elementos de datos de grupos a cualquier nivel entre 02 y 49.

Puede utilizar elementos de datos de grupos y los subordinados de los mismos de cuatro maneras:

Método 1.

Se puede hacer referencia al grupo entero como una sola variable del lenguaje principal en una sentencia de SQL:

```
EXEC SQL SELECT id, name, dept, job
  INTO :staff-record
  FROM staff WHERE id = 10 END-EXEC.
```

El precompilador convierte la referencia a staff-record en una lista de todos los elementos subordinados declarados dentro de staff-record, separados por comas. Cada elemento elemental se califica con los nombres de grupo de todos los niveles, a fin de evitar conflictos de denominación con otros elementos. Esto es equivalente al siguiente método.

Método 2.

La segunda manera de utilizar elementos de datos de grupos:

```
EXEC SQL SELECT id, name, dept, job
  INTO
  :staff-record.staff-id,
  :staff-record.staff-name,
  :staff-record.staff-info.staff-dept,
  :staff-record.staff-info.staff-job
  FROM staff WHERE id = 10 END-EXEC.
```

Nota: la referencia a staff-id se califica con su nombre de grupo utilizando el prefijo staff-record., y no staff-id de staff-record como se hace en COBOL puro.

Suponiendo que no existen otras variables del lenguaje principal que tengan los mismos nombres que los subordinados de staff-record, la sentencia anterior también se puede codificar como en el método 3, eliminando la calificación explícita de grupo.

Método 3.

Aquí, se hace referencia a los elementos subordinados de la forma típica de COBOL, sin calificarlos con su elemento de grupo concreto:

```
EXEC SQL SELECT id, name, dept, job
  INTO
  :staff-id,
  :staff-name,
  :staff-dept,
  :staff-job
  FROM staff WHERE id = 10 END-EXEC.
```

Como en COBOL puro, este método resulta aceptable para el precompilador siempre que un elemento subordinado determinado se pueda calificar de

forma exclusiva. Por ejemplo, si staff-job aparece más de una vez en un grupo, el precompilador emite un error indicando que se ha producido una referencia ambigua:

```
SQL0088N La variable del lenguaje principal "staff-job" es ambigua.
```

Método 4.

Para resolver la referencia ambigua, puede utilizar una calificación parcial del elemento subordinado, por ejemplo:

```
EXEC SQL SELECT id, name, dept, job
  INTO
  :staff-id,
  :staff-name,
  :staff-info.staff-dept,
  :staff-info.staff-job
  FROM staff WHERE id = 10 END-EXEC.
```

Puesto que una referencia a un solo elemento de grupo, como en el método 1, es equivalente a una lista de sus subordinados, separados por comas, existen casos en que este tipo de referencia conduce a un error. Por ejemplo:

```
EXEC SQL CONNECT TO :staff-record END-EXEC.
```

Aquí, la sentencia CONNECT espera una sola variable del lenguaje principal basada en caracteres. Proporcionando en su lugar el elemento de datos de grupos staff-record, la variable del lenguaje principal tiene como resultado el error de precompilación siguiente:

```
SQL0087N  La variable del lenguaje principal "staff-record" es una estructura
          que se utiliza donde no se permiten referencias a estructuras.
```

Otros usos de los elementos de grupos que pueden ocasionar que se produzca un error SQL0087N incluyen: PREPARE, EXECUTE IMMEDIATE, CALL, variables de indicador y referencias a SQLDA. Los grupos que sólo tienen un subordinado están permitidos en ambas situaciones, puesto que se trata de referencias a subordinados individuales, como en los métodos 2, 3 y 4 anteriores.

Tablas de indicadores en COBOL

El precompilador COBOL da soporte a la declaración de tablas de variables de indicador, que es conveniente utilizar con elementos de datos de grupos. Se declaran del modo siguiente:

```
01 <indicator-table-name>.
    05 <indicator-name> pic s9(4) comp-5
                        occurs <table-size> times.
```

Por ejemplo:

```
01 staff-indicator-table.
    05 staff-indicator pic s9(4) comp-5
                      occurs 7 times.
```

Esta tabla de indicadores se puede utilizar eficazmente con el primer formato de referencia de elementos de grupos indicado anteriormente:

```
EXEC SQL SELECT id, name, dept, job
  INTO :staff-record :staff-indicator
  FROM staff WHERE id = 10 END-EXEC.
```

Aquí, el precompilador detecta que `staff-indicator` se ha declarado como tabla de indicadores y la expande en referencias a indicadores individuales cuando procesa la sentencia de SQL. `staff-indicator`(1) se asocia a `staff-id` de `staff-record`, `staff-indicator`(2) se asocia a `staff-name` de `staff-record`, y así sucesivamente.

Nota: si en la tabla de indicadores existen `k` entradas de indicador más que subordinados tiene el elemento de datos (por ejemplo, si `staff-indicator` tiene 10 entradas, haciendo que `k`=6), se pasan por alto las `k` entradas de más que se encuentran al final de la tabla de indicadores. Del mismo modo, si hay `k` entradas de indicador menos que subordinados, los `k` últimos subordinados del elemento de grupo no tienen asociado ningún indicador. *Tenga en cuenta que puede hacer referencia a elementos individuales en una tabla de indicadores en una sentencia de SQL.*

Conceptos relacionados:

- "Variables de indicador en COBOL" en la página 246

REDEFINES en elementos de datos de grupos COBOL

Cuando declare variables del lenguaje principal, puede utilizar la cláusula REDEFINES. Si declara un miembro de un elemento de datos de grupo con la cláusula REDEFINES y se hace referencia a ese elemento de datos de grupo como un todo en una sentencia de SQL, los elementos subordinados que contienen la cláusula REDEFINES no se expanden. Por ejemplo:

```
01 foo.
 10 a pic s9(4) comp-5.
 10 a1 redefines a pic x(2).
 10 b pic x(10).
```

La referencia a `foo` en una sentencia de SQL es como sigue:

```
... INTO :foo ...
```

La sentencia anterior es equivalente a:

```
... INTO :foo.a, :foo.b ...
```

Es decir, el elemento subordinado a1, que se declara con la cláusula REDEFINES, no se expande automáticamente en estas situaciones. Si a1 es inequívoco, se puede hacer una referencia explícita a un subordinado que tenga una cláusula REDEFINES en una sentencia de SQL, del modo siguiente:

```
... INTO :foo.a1 ...
```

o

```
... INTO :a1 ...
```

Sección declare de SQL con variables del lenguaje principal para COBOL

A continuación se muestra un ejemplo de sección de declaración de SQL con una variable del lenguaje principal declarada para los tipos de datos SQL soportados.

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
*
  01 age        PIC S9(4) COMP-5.
  01 divis      PIC S9(9) COMP-5.
  01 salary     PIC S9(6)V9(3) COMP-3.
  01 bonus      USAGE IS COMP-1.
  01 wage       USAGE IS COMP-2.
  01 nm         PIC X(5).
  01 varchar.
     49 leng    PIC S9(4) COMP-5.
     49 strg    PIC X(14).
  01 longvchar.
     49 len     PIC S9(4) COMP-5.
     49 str     PIC X(6027).
  01 MY-CLOB USAGE IS SQL TYPE IS CLOB(1M).
  01 MY-CLOB-LOCATOR USAGE IS SQL TYPE IS CLOB-LOCATOR.
  01 MY-CLOB-FILE USAGE IS SQL TYPE IS CLOB-FILE.
  01 MY-BLOB USAGE IS SQL TYPE IS BLOB(1M).
  01 MY-BLOB-LOCATOR USAGE IS SQL TYPE IS BLOB-LOCATOR.
  01 MY-BLOB-FILE USAGE IS SQL TYPE IS BLOB-FILE.
  01 MY-DBCLOB USAGE IS SQL TYPE IS DBCLOB(1M).
  01 MY-DBCLOB-LOCATOR USAGE IS SQL TYPE IS DBCLOB-LOCATOR.
  01 MY-DBCLOB-FILE USAGE IS SQL TYPE IS DBCLOB-FILE.
  01 MY-PICTURE PIC G(16000) USAGE IS DISPLAY-1.
  01 dt         PIC X(10).
  01 tm         PIC X(8).
  01 tmstmp     PIC X(26).
  01 wage-ind   PIC S9(4) COMP-5.
*
EXEC SQL END DECLARE SECTION END-EXEC.
```

Consulta relacionada:

- “Tipos de datos de SQL soportados en COBOL” en la página 254

Consideraciones sobre tipos de datos para COBOL

Las secciones siguientes describen cómo se correlacionan los tipos de datos de SQL con tipos de datos de COBOL.

Tipos de datos de SQL soportados en COBOL

Determinados tipos de datos de COBOL predefinidos corresponden a tipos de columna. Sólo se pueden declarar como variables del lenguaje principal estos tipos de datos de COBOL.

La tabla siguiente muestra el equivalente en COBOL de cada tipo de columna. Cuando el precompilador encuentra una declaración de una variable del lenguaje principal, determina el valor del tipo de SQL apropiado. El gestor de bases de datos utiliza este valor para convertir los datos intercambiados entre la aplicación y él mismo.

No se reconocen todas las descripciones de datos posibles para las variables del lenguaje principal. Los elementos de datos COBOL deben ser coherentes con los descritos en la tabla siguiente. Si utiliza otros elementos de datos, se puede producir un error.

Nota: no existe soporte de variables del lenguaje principal para el tipo de datos DATALINK en ninguno de los lenguajes principales de DB2.

Tabla 15. Tipos de datos SQL correlacionados con declaraciones COBOL

Tipo de columna SQL[1]	Tipo de datos de COBOL	Descripción del tipo de columna SQL
SMALLINT (500 ó 501)	01 name PIC S9(4) COMP-5.	Entero con signo de 16 bits
INTEGER (496 ó 497)	01 name PIC S9(9) COMP-5.	Entero con signo de 32 bits
BIGINT (492 ó 493)	01 name PIC S9(18) COMP-5.	Entero con signo de 64 bits
DECIMAL(*p*,*s*)(484 ó 485)	01 name PIC S9(*m*)V9(*n*) COMP-3.	Decimal empaquetado
REAL[2] (480 ó 481)	01 name USAGE IS COMP-1.	Coma flotante de precisión simple
DOUBLE[3] (480 ó 481)	01 name USAGE IS COMP-2.	Coma flotante de precisión doble
CHAR(*n*) (452 ó 453)	01 name PIC X(*n*).	Serie de caracteres de longitud fija
VARCHAR(*n*) (448 ó 449)	01 name. 49 length PIC S9(4) COMP-5. 49 name PIC X(*n*). 1<=*n*<=32 672	Serie de caracteres de longitud variable

Tabla 15. Tipos de datos SQL correlacionados con declaraciones COBOL (continuación)

Tipo de columna SQL[1]	Tipo de datos de COBOL	Descripción del tipo de columna SQL
LONG VARCHAR (456 ó 457)	01 name. 49 length PIC S9(4) COMP-5. 49 data PIC X(*n*). 32 673<=*n*<=32 700	Serie de caracteres de longitud variable larga
CLOB(*n*) (408 ó 409)	01 MY-CLOB USAGE IS SQL TYPE IS CLOB(n). 1<=*n*<=2 147 483 647	Serie de caracteres de longitud variable y objeto grande
Variable de localizador CLOB[4] (964 ó 965)	01 MY-CLOB-LOCATOR USAGE IS SQL TYPE IS CLOB-LOCATOR.	Identifica las entidades CLOB que residen en el servidor
Variable de referencia de archivo CLOB[4] (920 ó 921)	01 MY-CLOB-FILE USAGE IS SQL TYPE IS CLOB-FILE.	Descriptor de archivo que contiene datos CLOB
BLOB(*n*) (404 ó 405)	01 MY-BLOB USAGE IS SQL TYPE IS BLOB(n). 1<=*n*<=2 147 483 647	Serie binaria de longitud variable y objeto grande
Variable de localizador BLOB[4] (960 ó 961)	01 MY-BLOB-LOCATOR USAGE IS SQL TYPE IS BLOB-LOCATOR.	Identifica las entidades BLOB que residen en el servidor
Variable de referencia de archivo BLOB[4] (916 ó 917)	01 MY-CLOB-FILE USAGE IS SQL TYPE IS CLOB-FILE.	Descriptor de archivo que contiene datos CLOB
DATE (384 ó 385)	01 identifier PIC X(10).	Serie de caracteres de 10 bytes
TIME (388 ó 389)	01 identifier PIC X(8).	Serie de caracteres de 8 bytes
TIMESTAMP (392 ó 393)	01 identifier PIC X(26).	Serie de caracteres de 26 bytes
Nota: los tipos de datos siguientes sólo están disponibles en el entorno DBCS.		
GRAPHIC(*n*) (468 ó 469)	01 name PIC G(*n*) DISPLAY-1.	Serie de caracteres de doble byte y longitud fija
VARGRAPHIC(*n*) (464 ó 465)	01 name. 49 length PIC S9(4) COMP-5. 49 name PIC G(*n*) DISPLAY-1. 1<=*n*<=16 336	Serie de caracteres de doble byte y longitud variable con indicador de longitud de serie de 2 bytes
LONG VARGRAPHIC (472 ó 473)	01 name. 49 length PIC S9(4) COMP-5. 49 name PIC G(*n*) DISPLAY-1. 16 337<=*n*<=16 350	Serie de caracteres de doble byte y longitud variable con indicador de longitud de serie de 2 bytes

Tabla 15. Tipos de datos SQL correlacionados con declaraciones COBOL (continuación)

Tipo de columna SQL[1]	Tipo de datos de COBOL	Descripción del tipo de columna SQL
DBCLOB(*n*) (412 ó 413)	01 MY-DBCLOB USAGE IS SQL TYPE IS DBCLOB(n). 1<=*n*<=1 073 741 823	Serie de caracteres de doble byte, longitud variable y objeto grande con indicador de longitud de serie de 4 bytes
Variable de localizador DBCLOB[4] (968 ó 969)	01 MY-DBCLOB-LOCATOR USAGE IS SQL TYPE IS DBCLOB-LOCATOR.	Identifica las entidades DBCLOB que residen en el servidor
Variable de referencia de archivos DBCLOB[4] (924 ó 925)	01 MY-DBCLOB-FILE USAGE IS SQL TYPE IS DBCLOB-FILE.	Descriptor de archivo que contiene datos DBCLOB

Notas:

1. El primer número que se encuentra bajo **Tipo de columna SQL** indica que no se proporciona una variable de indicador, y el segundo número indica que se proporciona una variable de indicador. Se necesita una variable de indicador para indicar los valores nulos (NULL) o para contener la longitud de una serie truncada. Éstos son los valores que aparecerán en el campo SQLTYPE del SQLDA para estos tipos de datos.
2. FLOAT(*n*) donde 0 < *n* < 25 es un sinónimo de REAL. La diferencia entre REAL y DOUBLE en el SQLDA es el valor de la longitud (4 ó 8).
3. Los tipos SQL siguientes son sinónimos de DOUBLE:
   - FLOAT
   - FLOAT(*n*) donde 24 < *n* < 54 es
   - DOUBLE PRECISION
4. Éste no es un tipo de columna, sino un tipo de variable del lenguaje principal.

A continuación mostramos normas adicionales para los tipos de datos COBOL soportados:

- PIC S9 y COMP-3/COMP-5 son necesarios cuando aparecen.
- Puede utilizar el número de nivel 77 en lugar de 01 para todos los tipos de columna, a excepción de VARCHAR, LONG VARCHAR, VARGRAPHIC, LONG VARGRAPHIC y todos los tipos de variable LOB.
- Cuando declare variables del lenguaje principal para tipos de columna DECIMAL(p,s), siga las normas siguientes. Consulte el ejemplo siguiente:

  ```
  01 identifier PIC S9(m)V9(n) COMP-3
  ```

  – Utilice V para indicar la coma decimal.
  – Los valores para *n* y *m* deben ser mayores o iguales a 1.
  – El valor para *n* + *m* no puede exceder de 31.
  – El valor para *s* es igual al valor de *n*.
  – El valor para *p* es igual al valor de *n* + *m*.
  – Los factores de repetición *(n)* y *(m)* son opcionales. Todos los ejemplos siguientes son válidos:

```
01 identifier PIC S9(3)V COMP-3
01 identifier PIC SV9(3) COMP-3
01 identifier PIC S9V COMP-3
01 identifier PIC SV9 COMP-3
```

  - Se puede utilizar PACKED-DECIMAL en lugar de COMP-3.
- El precompilador de COBOL *no* da soporte a matrices.

Conceptos relacionados:

- "Sección declare de SQL con variables del lenguaje principal para COBOL" en la página 253

Tipos de datos BINARY/COMP-4 de COBOL

El precompilador COBOL de DB2 da soporte al uso de los tipos de datos BINARY, COMP y COMP-4 dondequiera que estén permitidas las variables del lenguaje principal y los indicadores, siempre que el compilador COBOL de destino vea (o se pueda hacer que vea) los tipos de datos BINARY, COMP o COMP-4 como equivalentes al tipo de datos COMP-5. En esta publicación, estas variables del lenguaje principal e indicadores se muestran con el tipo COMP-5. Los compiladores de destino soportados por DB2 que tratan COMP, COMP-4, BINARY COMP y COMP-5 como equivalentes son:

- IBM COBOL Set para AIX
- Micro Focus COBOL para AIX

FOR BIT DATA en COBOL

Determinadas columnas de bases de datos se pueden declarar FOR BIT DATA. Estas columnas, que generalmente contienen caracteres, se utilizan para contener información binaria. Los tipos de datos CHAR(*n*), VARCHAR, LONG VARCHAR y BLOB son los tipos de variable del lenguaje principal de COBOL que pueden contener datos binarios. Utilice estos tipos de datos cuando trabaje con columnas que tengan el atributo FOR BIT DATA.

Consulta relacionada:

- "Tipos de datos de SQL soportados en COBOL" en la página 254

Variables SQLSTATE y SQLCODE en COBOL

Cuando se utiliza la opción de precompilación LANGLEVEL con el valor SQL92E, se pueden incluir las dos declaraciones siguientes como variables del lenguaje principal:

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 SQLSTATE PICTURE X(5).
01 SQLCODE  PICTURE S9(9) USAGE COMP.
.
```

```
.
.
EXEC SQL END DECLARE SECTION END-EXEC.
```

Si no se especifica ninguna de ellas, se supone la declaración SQLCODE durante el paso de precompilación. 01 también puede ser 77 y PICTURE puede ser PIC. Observe que, si se utiliza esta opción, no se debe especificar la sentencia INCLUDE SQLCA.

Para las aplicaciones formadas por varios archivos fuente, las declaraciones SQLCODE y SQLSTATE se pueden incluir en cada uno de los archivos fuente, tal como en el ejemplo anterior.

Consideraciones sobre EUC en japonés o chino tradicional y UCS-2 para COBOL

Los datos gráficos enviados desde una aplicación que se ejecuta bajo un juego de códigos eucJp o eucTW, o se conecta a una base de datos UCS-2, se identifican con el identificador de la página de códigos UCS-2. La aplicación debe convertir una serie de caracteres gráficos a UCS-2 antes de enviarla al servidor de base de datos. Del mismo modo, los datos gráficos recuperados de una base de datos UCS-2 por una aplicación, o de cualquier base de datos por una aplicación que se ejecute bajo una página de códigos EUC eucJP o eucTW, se codifican utilizando UCS-2. Esto requiere que la aplicación realice internamente una conversión de UCS-2 a la página de códigos de la aplicación, a menos que se vayan a presentar datos UCS-2 al usuario.

La aplicación es responsable de convertir a UCS-2 y desde UCS-2, puesto que esta conversión se debe llevar a cabo antes de copiar los datos al SQLDA o desde esta. DB2 Universal Database no suministra ninguna rutina de conversión que resulte accesible para la aplicación. En cambio, se deben utilizar las llamadas del sistema disponibles desde el sistema operativo. En el caso de una base de datos UCS-2, también puede considerar la posibilidad de utilizar las funciones escalares VARCHAR y VARGRAPHIC.

Conceptos relacionados:

- “Consideraciones sobre los juegos de códigos EUC y UCS-2 de japonés y chino tradicional” en la página 444

Consulta relacionada:

- “Función escalar VARCHAR” en el manual *Consulta de SQL, Volumen 1*
- “Función escalar VARGRAPHIC” en el manual *Consulta de SQL, Volumen 1*

COBOL orientado a objetos

Si utiliza COBOL orientado a objetos, debe tener en cuenta lo siguiente:

- Sólo pueden aparecer sentencias de SQL en el primer programa o clase de una unidad de compilación. Esta restricción se debe a que el precompilador inserta datos de trabajo temporales en la primera sección Working-Storage que encuentra.
- En un programa en COBOL orientado a objetos, cada clase que contenga sentencias de SQL debe tener una sección Working-Storage a nivel de clase, aunque esté vacía. Esta sección se utiliza para almacenar definiciones de datos generadas por el precompilador.

Capítulo 9. Programación en FORTRAN

Consideraciones sobre la programación en FORTRAN

En las secciones siguientes se explican las consideraciones especiales sobre la programación en el lenguaje principal. Se incluye información sobre las restricciones de lenguaje, los archivos include específicos del lenguaje principal, la incorporación de sentencias de SQL y los tipos de datos soportados para las variables del lenguaje principal.

Nota: soporte de FORTRAN estabilizado en DB2 Versión 5, y no hay ninguna mejora de FORTRAN planificada para el futuro. Por ejemplo, el precompilador FORTRAN no puede manejar identificadores de objetos SQL, como por ejemplo nombres de tabla, que tengan una longitud superior a 18 bytes. Para utilizar las características incorporadas en DB2 después de la Versión 5, como por ejemplo los nombres de tabla de longitud entre 19 y 128 bytes, debe escribir sus aplicaciones en un lenguaje que no sea FORTRAN.

Restricciones del lenguaje en FORTRAN

Las siguientes secciones describen las restricciones del lenguaje correspondientes a FORTRAN.

Llamada por referencia en FORTRAN

Algunos parámetros de las API requieren direcciones en lugar de valores en las variables de llamada. El gestor de bases de datos proporciona las API GET ADDRESS, DEREFERENCE ADDRESS y COPY MEMORY, que simplifican la posibilidad de que el usuario proporcione estos parámetros.

Consulta relacionada:

- "sqlgdref - Dereference Address" en el manual *Administrative API Reference*
- "sqlgaddr - Get Address" en el manual *Administrative API Reference*
- "sqlgmcpy - Copy Memory" en el manual *Administrative API Reference*

Líneas de depuración y de comentario en FORTRAN

Algunos compiladores de FORTRAN tratan las líneas que contienen una 'D' o una 'd' en la columna 1 como líneas condicionales. Estas líneas se pueden compilar para su depuración o se pueden tratar como comentarios. El precompilador siempre tratará las líneas que contienen una 'D' o una 'd' en la columna 1 como comentarios.

Consideraciones sobre la precompilación en FORTRAN

Los elementos siguientes afectan al proceso de precompilación:

- El precompilador sólo admite dígitos, blancos y caracteres de tabulación en las columnas 1-5 de las líneas de continuación.
- Las constantes Hollerith no se reciben soporte en archivos fuente `.sqf`.

Acceso a bases de datos de varias hebras en FORTRAN

FORTRAN no da soporte al acceso a bases de datos de varias hebras.

Archivos de entrada y salida para FORTRAN

Por omisión, el archivo de entrada tiene la extensión `.sqf`, pero si se utiliza la opción de precompilación TARGET, el archivo de entrada puede tener la extensión que desee el usuario.

Por omisión, el archivo de salida tiene la extensión `.f` en plataformas basadas en UNIX y la extensión `.for` en plataformas basadas en Windows; sin

embargo, puede utilizar la opción de precompilación OUTPUT para especificar un nuevo nombre y vía de acceso para el archivo fuente de salida modificado.

Consulta relacionada:

- "Mandato PRECOMPILE" en el manual *Consulta de mandatos*

Archivos include

Las siguientes secciones describen los archivos include correspondientes a FORTRAN.

Archivos include para FORTRAN

Los archivos include específicos del lenguaje principal para FORTRAN tienen la extensión `.f` en plataformas basadas en UNIX y la extensión `.for` en plataformas basadas en Windows. Puede utilizar los siguientes archivos include de FORTRAN en las aplicaciones.

SQL (sql.f) Este archivo incluye prototipos específicos del lenguaje para el vinculador, el precompilador y las API de recuperación de mensajes de error. También define constantes del sistema.

SQLAPREP (sqlaprep.f)
Este archivo contiene definiciones necesarias para escribir su propio precompilador.

SQLCA (sqlca_cn.f, sqlca_cs.f)
Este archivo define la estructura del Área de comunicaciones de SQL (SQLCA). El SQLCA contiene variables que utiliza el gestor de bases de datos para proporcionar a una aplicación información de error sobre la ejecución de sentencias de SQL y llamadas a API.

Se proporcionan dos archivos SQLCA para las aplicaciones FORTRAN. El valor por omisión, `sqlca_cs.f`, define la estructura SQLCA en un formato compatible con SQL de IBM. El archivo `sqlca_cn.f`, precompilado con la opción `SQLCA NONE`, define la estructura SQLCA para un mejor rendimiento.

SQLCA_92 (sqlca_92.f)
Este archivo contiene una versión que se ajusta a FIPS SQL92 Entry Level de la estructura Área de comunicaciones de SQL (SQL Communications Area (SQLCA)). Se debe incluir este archivo en lugar de los archivos `sqlca_cn.f` o `sqlca_cs.f` cuando se escriban aplicaciones DB2 que se ajusten al estándar FIPS SQL92 Entry Level. El precompilador de DB2

incluye automáticamente el archivo sqlca_92.f cuando la opción LANGLEVEL del precompilador se establece en SQL92E.

SQLCODES (sqlcodes.f)
Este archivo define constantes para el campo SQLCODE de la estructura SQLCA.

SQLDA (sqldact.f)
Este archivo define la estructura del Área de descriptor de SQL (SQLDA). El SQLDA se utiliza para pasar datos entre una aplicación y el gestor de bases de datos.

SQLEAU (sqleau.f)
Este archivo contiene definiciones de constantes y de estructuras necesarias para las API de auditoría de seguridad de DB2. Si utiliza estas API, tiene que incluir este archivo en el programa. Este archivo también contiene definiciones de constantes y de valores de palabras clave para los campos del registro de seguimiento de auditoría. Programas externos o de extracción de seguimiento de auditoría de proveedores pueden utilizar estas definiciones.

SQLENV (sqlenv.f)
Este archivo define llamadas específicas del lenguaje para las API del entorno de bases de datos y las estructuras, constantes y códigos de retorno correspondientes a dichas interfaces.

SQLE819A (sqle819a.f)
Si la página de códigos de la base de datos es 819 (ISO Latin-1), esta secuencia clasifica series de caracteres que no son FOR BIT DATA según la clasificación binaria CCSID 500 (EBCDIC internacional) del sistema principal. La API CREATE DATABASE utiliza este archivo.

SQLE819B (sqle819b.f)
Si la página de códigos de la base de datos es 819 (ISO Latin-1), esta secuencia clasifica series de caracteres que no son FOR BIT DATA según la clasificación binaria CCSID 037 (EBCDIC inglés de EE.UU.) del sistema principal. La API CREATE DATABASE utiliza este archivo.

SQLE850A (sqle850a.f)
Si la página de códigos de la base de datos es 850 (ASCII Latin-1), esta secuencia clasifica series de caracteres que no son FOR BIT DATA según la clasificación binaria CCSID 500 (EBCDIC internacional) del sistema principal. La API CREATE DATABASE utiliza este archivo.

SQLE850B (sqle850b.f)

Si la página de códigos de la base de datos es 850 (ASCII Latin-1), esta secuencia clasifica series de caracteres que no son FOR BIT DATA según la clasificación binaria CCSID 037 (EBCDIC inglés de EE.UU.) del sistema principal. La API CREATE DATABASE utiliza este archivo.

SQLE932A (sqle932a.f)

Si la página de códigos de la base de datos es 932 (ASCII japonés), esta secuencia clasifica series de caracteres que no son FOR BIT DATA según la clasificación binaria CCSID 5035 (EBCDIC japonés) del sistema principal. La API CREATE DATABASE utiliza este archivo.

SQLE932B (sqle932b.f)

Si la página de códigos de la base de datos es 932 (ASCII japonés), esta secuencia clasifica series de caracteres que no son FOR BIT DATA según la clasificación binaria CCSID 5026 (EBCDIC japonés) del sistema principal. La API CREATE DATABASE utiliza este archivo.

SQL1252A (sql1252a.f)

Si la página de códigos de la base de datos es 1252 (Windows Latin-1), esta secuencia clasifica series de caracteres que no son FOR BIT DATA según la clasificación binaria CCSID 500 (EBCDIC internacional) del sistema principal. La API CREATE DATABASE utiliza este archivo.

SQL1252B (sql1252b.f)

Si la página de códigos de la base de datos es 1252 (Windows Latin-1), esta secuencia clasifica series de caracteres que no son FOR BIT DATA según la clasificación binaria CCSID 037 (EBCDIC inglés de EE.UU.) del sistema principal. La API CREATE DATABASE utiliza este archivo.

SQLMON (sqlmon.f)

Este archivo define llamadas específicas del lenguaje para las API del supervisor del sistema de bases de datos y las estructuras, constantes y códigos de retorno correspondientes a dichas interfaces.

SQLSTATE (sqlstate.f)

Este archivo define constantes correspondientes al campo SQLSTATE de la estructura SQLCA.

SQLUTIL (sqlutil.f)

Este archivo define las llamadas específicas del lenguaje

correspondientes a las API de programas de utilidad y las estructuras, constantes y códigos necesarios para dichas interfaces.

Conceptos relacionados:

- "Archivos include en aplicaciones FORTRAN" en la página 266

Archivos include en aplicaciones FORTRAN

Existen dos métodos para incluir archivos: la sentencia EXEC SQL INCLUDE y la sentencia INCLUDE de FORTRAN. El precompilador pasará por alto las sentencias INCLUDE de FORTRAN y sólo procesará los archivos incluidos mediante la sentencia EXEC SQL.

Para localizar el archivo INCLUDE, el precompilador FORTRAN de DB2 busca en primer lugar en el directorio actual y, a continuación, en los directorios especificados por la variable de entorno DB2INCLUDE. Considere los ejemplos siguientes:

- `EXEC SQL INCLUDE payroll`

  Si el archivo especificado en la sentencia INCLUDE no está encerrado entre comillas, tal como en el ejemplo anterior, el precompilador busca `payroll.sqf` y luego `payroll.f` (`payroll.for` en plataformas basadas en Windows), en cada uno de los directorios que mira.

- `EXEC SQL INCLUDE 'pay/payroll.f'`

  Si el nombre de archivo está encerrado entre comillas, tal como en el caso anterior, no se añade ninguna extensión al nombre. (Para plataformas basadas en Windows, el archivo se especificaría como `'pay\payroll.for'`.)

  Si el nombre de archivo entrecomillado no contiene una vía de acceso absoluta, se utiliza el contenido de DB2INCLUDE para buscar el archivo, añadiéndole como prefijo la vía de acceso especificada en el nombre del archivo de INCLUDE. Por ejemplo, con DB2 para plataformas basadas en AIX, si DB2INCLUDE se establece en '`/disk2:myfiles/fortran`', el precompilador busca '`./pay/payroll.f`', luego '`/disk2/pay/payroll.f`' y finalmente '`./myfiles/cobol/pay/payroll.f`'. En los mensajes del precompilador se muestra la vía de acceso en que se encuentra realmente el archivo. En plataformas basadas en Windows, sustituya las barras inclinadas invertidas (`\`) por barras inclinadas y sustituya `'for'` por la extensión `'f'` en el ejemplo anterior.

Nota: el procesador de línea de mandatos de DB2 coloca en antememoria el valor de DB2INCLUDE. Para cambiar el valor de DB2INCLUDE después de haber emitido mandatos del CLP, entre el mandato TERMINATE, luego vuelva a conectar con la base de datos y realice una precompilación del modo habitual.

Conceptos relacionados:

- "DB2 registry and environment variables" en el manual *Administration Guide: Performance*

Consulta relacionada:

- "Archivos include para FORTRAN" en la página 263

Sentencias de SQL incorporado en FORTRAN

Las sentencias de SQL incorporado constan de los tres elementos siguientes:

Elemento	Sintaxis correcta en FORTRAN
Palabra clave	EXEC SQL
Serie de la sentencia	Cualquier sentencia de SQL válida con blancos como delimitadores
Terminador de la sentencia	Fin de la línea fuente.

El fin de la línea fuente sirve como terminador de sentencia. Si se continúa la línea, el terminador de sentencia es el fin de la última línea de continuación.

Por ejemplo:

```
EXEC SQL SELECT COL INTO :hostvar FROM TABLE
```

Se aplican las normas siguientes a las sentencias de SQL incorporado:

- Codifique las sentencias de SQL únicamente entre las columnas 7 y 72.
- Utilice comentarios de FORTRAN de línea completa, o comentarios de SQL, pero no utilice el carácter '!' de comentario de fin de línea de FORTRAN en las sentencias de SQL. Este carácter de comentario se puede utilizar en cualquier otra parte, incluso en las declaraciones de variables del lenguaje principal.
- Utilice blancos como delimitadores cuando codifique sentencias de SQL incorporado, aunque las sentencias de FORTRAN no requieren blancos como delimitadores.
- Utilice únicamente una sentencia de SQL para cada línea fuente en FORTRAN. Para las sentencias que necesitan más de una línea fuente, se aplican las normas habituales de continuación de FORTRAN. No divida el par de palabras clave EXEC SQL en distintas líneas.
- Están permitidos los comentarios de SQL en cualquier línea que forme parte de una sentencia de SQL incorporado. Estos comentarios no están permitidos en las sentencias que se ejecutan de forma dinámica. El formato de un comentario de SQL consiste en un guión doble (--) seguido de una serie compuesta por cero o más caracteres y terminada por un fin de línea.

- Los comentarios FORTRAN están permitidos *casi* en cualquier lugar de una sentencia de SQL incorporada. Las excepciones son:
  - No se permiten comentarios entre EXEC y SQL.
  - No se permiten comentarios en las sentencias que se ejecutan dinámicamente.
  - La ampliación de utilizar ! para codificar un comentario FORTRAN al final de una línea no recibe soporte dentro de una sentencia de SQL incorporado.
- Utilice una notación exponencial cuando especifique una constante real en las sentencias de SQL. El gestor de bases de datos interpreta una serie de dígitos con una coma decimal en una sentencia de SQL como una constante decimal, no como una constante real.
- Los números de sentencia no son válidos en las sentencias de SQL que preceden a la primera sentencia FORTRAN ejecutable. Si una sentencia de SQL tiene asociado un número de sentencia, el precompilador genera una sentencia CONTINUE etiquetada que precede directamente a la sentencia de SQL.
- Cuando haga referencia a variables del lenguaje principal dentro de una sentencia de SQL, utilícelas exactamente tal como se han declarado.
- La sustitución de caracteres de espacio en blanco, como caracteres de fin de línea y de tabulación, se produce del siguiente modo:
  - Cuando aparecen fuera de las comillas (pero dentro de sentencias de SQL), los caracteres de fin de línea y tabuladores se sustituyen por un solo espacio.
  - Si aparecen dentro de las comillas, los caracteres de fin de línea desaparecen, siempre que se continúe correctamente la serie para un programa FORTRAN. Los tabuladores no se modifican.

  Observe que los caracteres reales utilizados como fin de línea y tabulador varían en función de la plataforma. Por ejemplo, las plataformas basadas en Windows utilizan Retorno de carro/Salto de línea como fin de línea, mientras que las plataformas basadas en UNIX sólo utilizan un Salto de línea.

Consulta relacionada:

- Apéndice A, “Sentencias de SQL soportadas” en la página 521

Variables del lenguaje principal en FORTRAN

Las secciones siguientes describen cómo declarar y utilizar variables del lenguaje principal en programas FORTRAN.

Variables del lenguaje principal en FORTRAN

Las variables del lenguaje principal son variables del lenguaje FORTRAN a las que se hace referencia en sentencias de SQL. Permiten que una aplicación pase datos de entrada al gestor de bases de datos y reciba datos de salida de éste. Una vez que se ha precompilado la aplicación, el compilador utiliza las variables del lenguaje principal como cualquier otra variable de FORTRAN.

Conceptos relacionados:

- "Nombres de variables del lenguaje principal en FORTRAN" en la página 269
- "Declaraciones de variables del lenguaje principal en FORTRAN" en la página 270
- "Variables de indicador en FORTRAN" en la página 273

Consulta relacionada:

- "Sintaxis de las variables numéricas del lenguaje principal en FORTRAN" en la página 271
- "Sintaxis de las variables de tipo carácter del lenguaje principal en FORTRAN" en la página 271
- "Sintaxis de las variables del lenguaje principal de objeto grande (LOB) en FORTRAN" en la página 273
- "Sintaxis de las variables del lenguaje principal de localizador de objeto grande (LOB) en FORTRAN" en la página 274
- "Sintaxis de las variables del lenguaje principal de referencia de archivos en FORTRAN" en la página 275

Nombres de variables del lenguaje principal en FORTRAN

El precompilador SQL identifica las variables del lenguaje principal por el nombre declarado para éstas. Se aplican las sugerencias siguientes:

- Especifique nombres de variables con una longitud máxima de 255 caracteres.
- Empiece los nombres de variables con prefijos que no sean `SQL`, `sql`, `DB2` ni `db2`, que están reservados para uso del sistema.

Conceptos relacionados:

- "Declaraciones de variables del lenguaje principal en FORTRAN" en la página 270

Consulta relacionada:

- "Sintaxis de las variables numéricas del lenguaje principal en FORTRAN" en la página 271

- “Sintaxis de las variables de tipo carácter del lenguaje principal en FORTRAN” en la página 271
- “Sintaxis de las variables del lenguaje principal de objeto grande (LOB) en FORTRAN” en la página 273
- “Sintaxis de las variables del lenguaje principal de localizador de objeto grande (LOB) en FORTRAN” en la página 274
- “Sintaxis de las variables del lenguaje principal de referencia de archivos en FORTRAN” en la página 275

Declaraciones de variables del lenguaje principal en FORTRAN

Se debe utilizar una sección de declaración de SQL para identificar las declaraciones de variables del lenguaje principal. Esto alerta al precompilador acerca de las variables del lenguaje principal a las que se puede hacer referencia en sentencias de SQL posteriores.

El precompilador FORTRAN sólo reconoce un subconjunto de declaraciones de FORTRAN válidas como declaraciones de variables del lenguaje principal válidas. Estas declaraciones definen variables numéricas o de tipo carácter. Una variable numérica del lenguaje principal se puede utilizar como variable de entrada o de salida para cualquier valor numérico de entrada o salida de SQL. Una variable del lenguaje principal de tipo carácter se puede utilizar como variable de entrada o de salida para cualquier valor de tipo carácter, de fecha, de hora o de indicación de la hora, de entrada o salida de SQL. El programador se tiene que asegurar de que las variables de salida sean lo suficientemente largas como para contener los valores que van a recibir.

Tareas relacionadas:

- “Declaración de variables del lenguaje principal de tipo estructurado” en el manual *Guía de desarrollo de aplicaciones: Programación de aplicaciones de servidor*

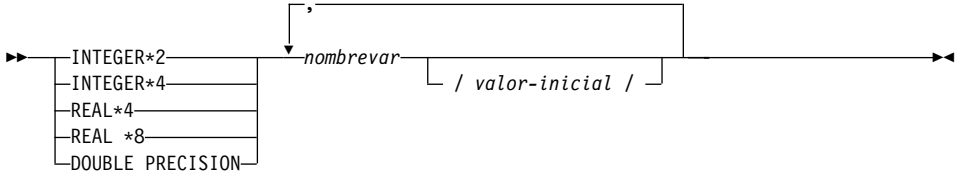
Consulta relacionada:

- “Sintaxis de las variables numéricas del lenguaje principal en FORTRAN” en la página 271
- “Sintaxis de las variables de tipo carácter del lenguaje principal en FORTRAN” en la página 271
- “Sintaxis de las variables del lenguaje principal de objeto grande (LOB) en FORTRAN” en la página 273
- “Sintaxis de las variables del lenguaje principal de localizador de objeto grande (LOB) en FORTRAN” en la página 274
- “Sintaxis de las variables del lenguaje principal de referencia de archivos en FORTRAN” en la página 275

Sintaxis de las variables numéricas del lenguaje principal en FORTRAN

A continuación se muestra la sintaxis de las variables numéricas del lenguaje principal en FORTRAN.

Sintaxis de las variables numéricas del lenguaje principal en FORTRAN



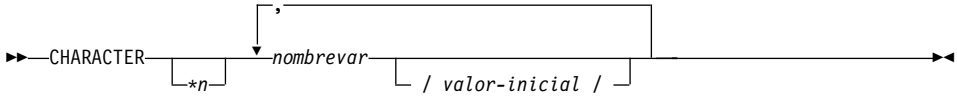
Consideraciones sobre las variables numéricas del lenguaje principal:

1. REAL*8 y DOUBLE PRECISION son equivalentes.
2. Utilice una E en lugar de una D como indicador de exponente para las constantes REAL*8.

Sintaxis de las variables de tipo carácter del lenguaje principal en FORTRAN

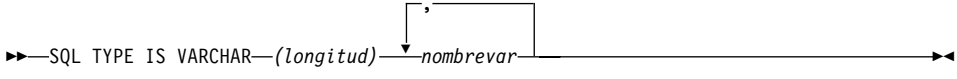
A continuación se muestra la sintaxis de las variables de tipo carácter de longitud fija del lenguaje principal.

Sintaxis de las variables del lenguaje principal de tipo carácter en FORTRAN: Longitud fija



A continuación se muestra la sintaxis de las variables de tipo carácter de longitud variable del lenguaje principal.

Longitud variable



Consideraciones sobre las variables del lenguaje principal de tipo carácter:

1. **n* tiene un valor máximo de 254.
2. Si la longitud está entre 1 y 32 672, ambos inclusive, la variable del lenguaje principal es de tipo VARCHAR(SQLTYPE 448).

3. Si la longitud está entre 32 673 y 32 700, ambos inclusive, la variable del lenguaje principal es de tipo LONG VARCHAR(SQLTYPE 456).
4. No está permitida la inicialización de variables del lenguaje principal VARCHAR y LONG VARCHAR dentro de la declaración.

Ejemplo de VARCHAR:

La declaración:

```
sql type is varchar(1000) my_varchar
```

Tiene como resultado la generación de la estructura siguiente:

```
character    my_varchar(1000+2)
integer*2    my_varchar_length
character    my_varchar_data(1000)
equivalence( my_varchar(1),
+            my_varchar_length )
equivalence( my_varchar(3),
+            my_varchar_data )
```

La aplicación puede manipular tanto `my_varchar_length` como `my_varchar_data`; por ejemplo, para establecer o examinar el contenido de la variable del lenguaje principal. El nombre base (en este caso, `my_varchar`), se utiliza en las sentencias de SQL para hacer referencia a la VARCHAR como un todo.

Ejemplo de LONG VARCHAR:

La declaración:

```
sql type is varchar(10000) my_lvarchar
```

Tiene como resultado la generación de la estructura siguiente:

```
character    my_lvarchar(10000+2)
integer*2    my_lvarchar_length
character    my_lvarchar_data(10000)
equivalence( my_lvarchar(1),
+            my_lvarchar_length )
equivalence( my_lvarchar(3),
+            my_lvarchar_data )
```

La aplicación puede manipular tanto `my_lvarchar_length` como `my_lvarchar_data`; por ejemplo, para establecer o examinar el contenido de la variable del lenguaje principal. El nombre base (en este caso, `my_lvarchar`) se utiliza en las sentencias de SQL para hacer referencia a la LONG VARCHAR como un todo.

Nota: en una sentencia CONNECT, como por ejemplo la que se muestra a continuación, a las variables del lenguaje principal de serie de caracteres en FORTRAN dbname y userid se les eliminarán los blancos de cola antes del proceso.

```
EXEC SQL CONNECT TO :dbname USER :userid USING :passwd
```

No obstante, puesto que los blancos pueden ser significativos en las contraseñas, debe declarar las variables del lenguaje principal para contraseñas como VARCHAR, y hacer que el campo de longitud establecido refleje la longitud real de la contraseña:

```
EXEC SQL BEGIN DECLARE SECTION
  character*8 dbname, userid
  sql type is varchar(18) passwd
EXEC SQL END DECLARE SECTION
character*18 passwd_string
equivalence(passwd_data,passwd_string)
dbname = 'sample'
userid = 'userid'
passwd_length= 8
passwd_string = 'password'
EXEC SQL CONNECT TO :dbname USER :userid USING :passwd
```

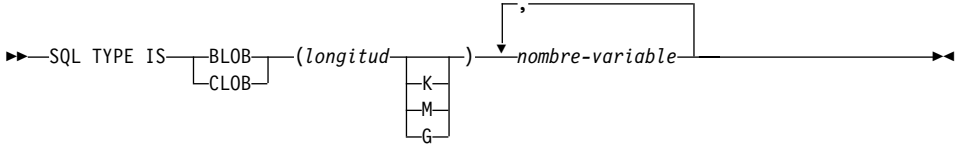
Variables de indicador en FORTRAN

Las variables de indicador se deben declarar con tipo de datos INTEGER*2.

Sintaxis de las variables del lenguaje principal de objeto grande (LOB) en FORTRAN

A continuación se muestra la sintaxis para declarar variables del lenguaje principal de objeto grande (LOB) en FORTRAN.

Sintaxis de las variables del lenguaje principal de objeto grande (LOB) en FORTRAN



Consideraciones sobre las variables del lenguaje principal LOB:

1. Los tipos GRAPHIC no se soportan en FORTRAN.
2. SQL TYPE IS, BLOB, CLOB, K, M, G se pueden especificar en mayúsculas, en minúsculas o en una mezcla de ambas.
3. Para BLOB y CLOB 1 <= longitud-lob <= 2 147 483 647.
4. No se permite la inicialización de un LOB dentro de una declaración LOB.

5. El nombre de la variable del lenguaje principal es el prefijo de 'longitud? y 'datos' en el código generado por el precompilador.

Ejemplo de BLOB:

La declaración:

```
sql type is blob(2m) my_blob
```

Tiene como resultado la generación de la estructura siguiente:

```
 character    my_blob(2097152+4)
 integer*4    my_blob_length
 character    my_blob_data(2097152)
 equivalence( my_blob(1),
+             my_blob_length )
 equivalence( my_blob(5),
+             my_blob_data )
```

Ejemplo de CLOB:

La declaración:

```
sql type is clob(125m) my_clob
```

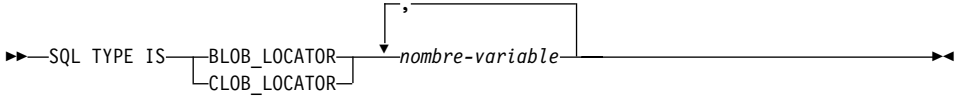
Tiene como resultado la generación de la estructura siguiente:

```
 character    my_clob(131072000+4)
 integer*4    my_clob_length
 character    my_clob_data(131072000)
 equivalence( my_clob(1),
+             my_clob_length )
 equivalence( my_clob(5),
+             my_clob_data )
```

Sintaxis de las variables del lenguaje principal de localizador de objeto grande (LOB) en FORTRAN

A continuación se muestra la sintaxis para declarar variables del lenguaje principal de localizador de objeto grande (LOB) en FORTRAN.

Sintaxis de las variables del lenguaje principal de localizador de objeto grande (LOB) en FORTRAN



Consideraciones sobre las variables del lenguaje principal de localizador de LOB:

1. Los tipos GRAPHIC no se soportan en FORTRAN.

2. SQL TYPE IS, BLOB_LOCATOR, CLOB_LOCATOR se pueden especificar en mayúsculas, en minúsculas o en una mezcla de ambas.
3. No se permite la inicialización de localizadores.

Ejemplo de localizador de CLOB (El localizador de BLOB es parecido):

La declaración:

```
SQL TYPE IS CLOB_LOCATOR my_locator
```

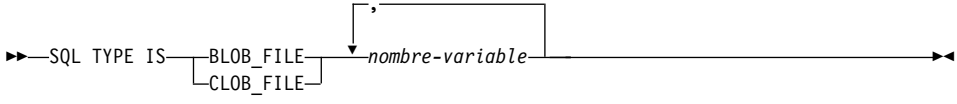
Tiene como resultado la generación de la declaración siguiente:

```
integer*4 my_locator
```

Sintaxis de las variables del lenguaje principal de referencia de archivos en FORTRAN

A continuación se muestra la sintaxis para declarar variables del lenguaje principal de referencia de archivos en FORTRAN.

Sintaxis de las variables del lenguaje principal de referencia de archivos en FORTRAN



Consideraciones sobre las variables del lenguaje principal de referencia de archivos:

1. Los tipos GRAPHIC no se soportan en FORTRAN.
2. SQL TYPE IS, BLOB-FILE, CLOB-FILE se pueden especificar en mayúsculas, en minúsculas o en una mezcla de ambas.

Ejemplo de una variable de referencia de archivos BLOB (La variable de referencia de archivos CLOB es parecida):

```
SQL TYPE IS BLOB_FILE my_file
```

Tiene como resultado la generación de la declaración siguiente:

```
 character     my_file(267)
 integer*4     my_file_name_length
 integer*4     my_file_data_length
 integer*4     my_file_file_options
 character*255 my_file_name
 equivalence(  my_file(1),
+              my_file_name_length )
 equivalence(  my_file(5),
+              my_file_data_length )
```

```
      equivalence(  my_file(9),
     +              my_file_file_options )
      equivalence(  my_file(13),
     +              my_file_name )
```

Sección declare de SQL con variables del lenguaje principal para FORTRAN

A continuación se muestra un ejemplo de sección de declaración de SQL con una variable del lenguaje principal declarada para cada tipo de datos soportado:

```
EXEC SQL BEGIN DECLARE SECTION
  INTEGER*2     AGE  /26/
  INTEGER*4     DEPT
  REAL*4        BONUS
  REAL*8        SALARY
  CHARACTER     MI
  CHARACTER*112 ADDRESS
  SQL TYPE IS VARCHAR (512) DESCRIPTION
  SQL TYPE IS VARCHAR (32000) COMMENTS
  SQL TYPE IS CLOB (1M) CHAPTER
  SQL TYPE IS CLOB_LOCATOR CHAPLOC
  SQL TYPE IS CLOB_FILE  CHAPFL
  SQL TYPE IS BLOB (1M) VIDEO
  SQL TYPE IS BLOB_LOCATOR VIDLOC
  SQL TYPE IS BLOB_FILE VIDFL
  CHARACTER*10  DATE
  CHARACTER*8   TIME
  CHARACTER*26  TIMESTAMP
  INTEGER*2     WAGE_IND
EXEC SQL END DECLARE SECTION
```

Consulta relacionada:

- "Tipos de datos SQL soportados en FORTRAN" en la página 276

Tipos de datos SQL soportados en FORTRAN

Determinados tipos de datos de FORTRAN predefinidos corresponden a tipos de columna del gestor de bases de datos. Sólo se pueden declarar como variables del lenguaje principal estos tipos de datos de FORTRAN.

La tabla siguiente muestra el equivalente en FORTRAN de cada tipo de columna. Cuando el precompilador encuentra una declaración de una variable del lenguaje principal, determina el valor del tipo de SQL apropiado. El gestor de bases de datos utiliza este valor para convertir los datos intercambiados entre la aplicación y él mismo.

Nota: no existe soporte de variables del lenguaje principal para el tipo de datos DATALINK en ninguno de los lenguajes principales de DB2.

Tabla 16. Tipos de datos SQL correlacionados con declaraciones FORTRAN

Tipo de columna SQL[1]	**Tipo de datos FORTRAN**	**Descripción del tipo de columna SQL**
SMALLINT (500 ó 501)	INTEGER*2	Entero con signo de 16 bits
INTEGER (496 ó 497)	INTEGER*4	Entero con signo de 32 bits
REAL[2] (480 ó 481)	REAL*4	Coma flotante de precisión simple
DOUBLE[3] (480 ó 481)	REAL*8	Coma flotante de precisión doble
DECIMAL(*p*,*s*)(484 ó 485)	No existe un equivalente exacto; utilice REAL*8	Decimal empaquetado
CHAR(*n*) (452 ó 453)	CHARACTER**n*	Serie de caracteres de longitud fija con longitud *n*, donde *n* va de 1 a 254
VARCHAR(*n*) (448 ó 449)	SQL TYPE IS VARCHAR(*n*), donde *n* va de 1 a 32 672	Serie de caracteres de longitud variable
LONG VARCHAR (456 ó 457)	SQL TYPE IS VARCHAR(*n*), donde *n* va de 32 673 a 32 700	Serie de caracteres de longitud variable larga
CLOB(*n*) (408 ó 409)	SQL TYPE IS CLOB (n) donde *n* va de 1 a 2 147 483 647	Serie de caracteres de longitud variable y objeto grande
Variable de localizador CLOB[4] (964 ó 965)	SQL TYPE IS CLOB_LOCATOR	Identifica las entidades CLOB que residen en el servidor
Variable de referencia de archivo CLOB[4] (920 ó 921)	SQL TYPE IS CLOB_FILE	Descriptor de archivo que contiene datos CLOB
BLOB(*n*) (404 ó 405)	SQL TYPE IS BLOB (n) donde *n* va de 1 a 2 147 483 647	Serie binaria de longitud variable y objeto grande
Variable de localizador BLOB[4] (960 ó 961)	SQL TYPE IS BLOB_LOCATOR	Identifica las entidades BLOB del servidor
Variable de referencia de archivo BLOB[4] (916 ó 917)	SQL TYPE IS BLOB_FILE	Descriptor del archivo que contiene datos BLOB
DATE (384 ó 385)	CHARACTER*10	Serie de caracteres de 10 bytes
TIME (388 ó 389)	CHARACTER*8	Serie de caracteres de 8 bytes
TIMESTAMP (392 ó 393)	CHARACTER*26	Serie de caracteres de 26 bytes

Tabla 16. Tipos de datos SQL correlacionados con declaraciones FORTRAN (continuación)

Tipo de columna SQL[1]	Tipo de datos FORTRAN	Descripción del tipo de columna SQL

Notas:

1. El primer número que se encuentra bajo **Tipo de columna SQL** indica que no se proporciona una variable de indicador, y el segundo número indica que se proporciona una variable de indicador. Se necesita una variable de indicador para indicar los valores nulos (NULL) o para contener la longitud de una serie truncada. Éstos son los valores que aparecerán en el campo SQLTYPE del SQLDA para estos tipos de datos.
2. FLOAT(*n*) donde 0 < *n* < 25 es un sinónimo de REAL. La diferencia entre REAL y DOUBLE en el SQLDA es el valor de la longitud (4 ó 8).
3. Los tipos SQL siguientes son sinónimos de DOUBLE:
   - FLOAT
   - FLOAT(*n*) donde 24 < *n* < 54 es
   - DOUBLE PRECISION
4. Éste no es un tipo de columna, sino un tipo de variable del lenguaje principal.

A continuación mostramos una norma adicional para los tipos de datos FORTRAN soportados:

- Puede definir sentencias de SQL dinámicas que contengan más de 254 caracteres utilizando las variables del lenguaje principal VARCHAR, LONG VARCHAR o CLOB.

Conceptos relacionados:

- "Sección declare de SQL con variables del lenguaje principal para FORTRAN" en la página 276

Consideraciones sobre juegos de caracteres de varios bytes en FORTRAN

En FORTRAN no se da soporte a ningún tipo de datos de variables del lenguaje principal de gráficos (de varios bytes). Sólo se da soporte a variables del lenguaje principal de tipo carácter mixtas mediante el tipo de datos `character`. Es posible crear una SQLDA de usuario que contenga datos gráficos.

Consideraciones sobre EUC en japonés o chino tradicional y UCS-2 para FORTRAN

Los datos gráficos enviados desde una aplicación que se ejecuta bajo un juego de códigos eucJp o eucTW, o se conecta a una base de datos UCS-2, se identifican con el identificador de la página de códigos UCS-2. La aplicación debe convertir una serie de caracteres gráficos a UCS-2 antes de enviarla al servidor de base de datos. Del mismo modo, los datos gráficos recuperados de una base de datos UCS-2 por una aplicación, o de cualquier base de datos por una aplicación que se ejecute bajo una página de códigos EUC eucJP o

eucTW, se codifican utilizando UCS-2. Esto requiere que la aplicación realice internamente una conversión de UCS-2 a la página de códigos de la aplicación, a menos que se vayan a presentar datos UCS-2 al usuario.

La aplicación es responsable de convertir a UCS-2 y desde UCS-2, puesto que esta conversión se debe llevar a cabo antes de copiar los datos al SQLDA o desde esta. DB2 Universal Database no suministra ninguna rutina de conversión que resulte accesible para la aplicación. En cambio, se deben utilizar las llamadas del sistema disponibles desde el sistema operativo. En el caso de una base de datos UCS-2, también puede considerar la posibilidad de utilizar las funciones escalares VARCHAR y VARGRAPHIC.

Conceptos relacionados:

- "Consideraciones sobre los juegos de códigos EUC y UCS-2 de japonés y chino tradicional" en la página 444

Consulta relacionada:

- "Función escalar VARCHAR" en el manual *Consulta de SQL, Volumen 1*
- "Función escalar VARGRAPHIC" en el manual *Consulta de SQL, Volumen 1*

Variables SQLSTATE y SQLCODE en FORTRAN

Cuando se utiliza la opción de precompilación LANGLEVEL con el valor SQL92E, se pueden incluir las dos declaraciones siguientes como variables del lenguaje principal:

```
EXEC SQL BEGIN DECLARE SECTION;
  CHARACTER*5 SQLSTATE
  INTEGER     SQLCOD
  .
  .
  .
EXEC SQL END DECLARE SECTION
```

Si no es especifica ninguna de ellas, se supone la declaración SQLCOD durante el paso de precompilación. La variable denominada SQLSTATE también puede ser SQLSTA. Observe que, si se utiliza esta opción, no se debe especificar la sentencia INCLUDE SQLCA.

Para aplicaciones que contienen varios archivos fuente, se pueden incluir las declaraciones de SQLCOD y SQLSTATE en cada archivo fuente, tal como se ha mostrado anteriormente.

Consulta relacionada:

- "Mandato PRECOMPILE" en el manual *Consulta de mandatos*

Parte 3. Java

Capítulo 10. Programación en Java

Consideraciones sobre la programación en Java

DB2 Universal Database implanta dos API de programación en Java basadas en estándares: Java Database Connectivity (JDBC) y SQL incorporado para Java (SQLj). Este capítulo contiene una visión general de la programación JDBC y SQLj, pero se centra en los aspectos específicos de DB2. Consulte el sitio Web de Java de DB2 Universal Database para ver enlaces con las especificaciones JDBC y SQLj.

Consulta relacionada:

- "Programas de ejemplo JDBC" en el manual *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Programas de ejemplo SQLJ" en el manual *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

JDBC y SQLj

Las secciones siguientes comparan JDBC y SQLj y describen la interoperatividad y las sesiones compartidas entre JDBC y SQLj.

Comparación entre SQLj y JDBC

La API JDBC le permite escribir programas Java que realizan llamadas de SQL dinámico a bases de datos. Las aplicaciones SQLj utilizan JDBC como base para tareas como conectar con las bases de datos y manejar errores de SQL, pero también pueden contener sentencias de SQL estático incorporado en los archivos fuente SQLj. Debe convertir un archivo fuente SQLj con el conversor de SQLj antes de compilar el código fuente Java resultante.

Interoperatividad entre JDBC y SQLj

El lenguaje SQLj proporciona soporte directo para operaciones de SQL estático conocidas en el momento en que se escribe el programa. Si parte de una determinada sentencia de SQL, o toda ella, no se puede determinar hasta el tiempo de ejecución, se trata de una operación dinámica. Para realizar operaciones de SQL dinámico desde un programa SQLj, utilice JDBC. Un objeto ConnectionContext contiene un objeto Conexión JDBC, que se puede utilizar para crear objetos de Sentencia de JDBC necesarios para operaciones de SQL dinámico.

Cada clase de SQLj ConnectionContext incluye un constructor que toma como argumento una Conexión JDBC. Este constructor se utiliza para crear una instancia de contexto de conexión SQLj que comparte su conexión a la base de datos subyacente con la de la conexión JDBC.

Cada instancia de SQLj ConnectionContext contiene un método `getConnection()` que devuelve una instancia Conexión JDBC. La Conexión JDBC devuelta comparte la conexión a la base de datos subyacente con el contexto de conexión SQLj. Se puede utilizar para realizar operaciones de SQL dinámico tal como se describe en la API de JDBC.

Conceptos relacionados:

- "Sesiones compartidas entre JDBC y SQLj" en la página 285
- "Gestión de recursos de conexión en Java" en la página 286

Sesiones compartidas entre JDBC y SQLj

Los métodos de interoperatividad entre JDBC y SQLj proporcionan una conversión entre las abstracciones de conexión que se utilizan en SQLj y las que se utilizan en JDBC. Ambas abstracciones comparten la misma sesión de base de datos, es decir, la conexión a la base de datos subyacente. Por lo tanto, las llamadas a métodos que afectan al estado de la sesión en un objeto también se verán reflejadas en el otro objeto, puesto que es realmente la sesión compartida subyacente la que se ve afectada.

JDBC define los valores por omisión para el estado de sesión de conexiones recién creadas. En la mayoría de los casos, SQLj adopta estos valores por omisión. Sin embargo, mientras que una conexión JDBC recién creada tiene la modalidad de confirmación automática activada por omisión, un contexto de conexión SQLj requiere que la modalidad de confirmación automática se especifique de forma explícita durante la construcción.

Conceptos relacionados:

- "Interoperatividad entre JDBC y SQLj" en la página 284
- "Gestión de recursos de conexión en Java" en la página 286

Ventajas de Java sobre otros lenguajes

Los lenguajes de programación que contienen SQL incorporado se denominan lenguajes principales. Java difiere de los lenguajes principales tradicionales C, COBOL y FORTRAN en formas que afectan significativamente al modo en que incorpora SQL:

- SQLj y JDBC son estándares abiertos que le permiten transportar fácilmente aplicaciones SQLj o JDBC desde otros sistemas de bases de datos compatibles con los estándares a DB2 Universal Database.
- Todos los tipos Java que representan datos compuestos y datos de tamaños variables tienen un valor distintivo, `null`, que se puede utilizar para

representar el estado NULL de SQL, lo que ofrece a los programas Java una alternativa a los indicadores NULL que constituyen un elemento fijo de otros lenguajes principales.

- Java está diseñado para dar soporte a programas que son portables de forma automática y heterogénea (también denominados ″super portables″ o simplemente ″descargables″). Junto con el sistema de clases e interfaces de tipo de Java, esta función habilita software de componentes. En particular, un conversor SQLj escrito en Java puede llamar a componentes especializados de proveedores de bases de datos para aprovechar las funciones de bases de datos existentes como autorización, comprobación de esquema, comprobación de tipo, funciones de transacción y recuperación, y para generar código optimizado para bases de datos específicas.
- Java está diseñado para su portabilidad binaria en redes heterogéneas, lo que permite la portabilidad binaria para aplicaciones de bases de datos que utilizan SQL estático.

Seguridad de SQL en Java

Por omisión, un programa JDBC ejecuta sentencias de SQL con privilegios asignados a la persona que ejecuta el programa. Por el contrario, un programa SQLj ejecuta sentencias de SQL con los privilegios asignados a la persona que ha creado el paquete de bases de datos.

Gestión de recursos de conexión en Java

Cuando se llama al método `close()` de una instancia del contexto de conexión, la instancia de conexión JDBC asociada y la conexión a la base de datos subyacente se cierran. Puesto que los contextos de conexión pueden compartir la conexión a la base de datos subyacente con otros contextos de conexión y/o con otras conexiones JDBC, es posible que no desee cerrar la conexión a la base de datos subyacente cuando se cierra un contexto de conexión. Puede que un programador desee liberar los recursos que mantiene el contexto de conexión (por ejemplo, descriptores de contexto de sentencias), sin cerrar realmente la conexión a la base de datos subyacente. Para ello, las clases de contexto de conexión también dan soporte a un método `close()`, que toma un argumento Booleano que indica si se debe o no cerrar la conexión a base de datos subyacente: la constante `CLOSE_CONNECTION` si se debe cerrar la conexión a base de datos y `KEEP_CONNECTION` si se debe mantener. La variante de `close()` que no toma ningún argumento es un sistema taquigráfico para llamar a `close(CLOSE_CONNECTION)`.

Si una instancia de contexto de conexión no se cierra de forma explícita antes de que se recopile como elemento desechable, el método finalice de este contexto de conexión llama a `close(KEEP_CONNECTION)`. Esto permite que el

proceso normal de recopilación de basura reclame recursos relacionados con la conexión mientras se mantiene la conexión a la base de datos subyacente para otros objetos JDBC y SQLj que la puedan estar utilizando. Observe que si ningún otro objeto JDBC o SQLj está utilizando la conexión, la conexión a base de datos se cierra y el proceso de recopilación de basura la reclama.

Tanto los objetos de contexto de conexión SQLj como los objetos de conexión JDBC responden al método `close()`. Cuando se escribe un programa SQLj, resulta suficiente llamar al método `close()` sólo en el objeto de contexto de conexión; al cerrar el contexto de conexión también se cierra la conexión JDBC asociada al mismo. Sin embargo, no es suficiente cerrar sólo la conexión JDBC que devuelve el método `getConnection()` de un contexto de conexión: el método `close()` de una conexión JDBC no hace que se cierre el contexto de conexión que la contiene, y por lo tanto los recursos que mantiene el contexto de conexión no se liberan hasta que se recopilan como basura.

El método `isClosed()` de un contexto de conexión devuelve `true` si se ha llamado a cualquier variante del método `close()` en la instancia del contexto de conexión. Si `isClosed()` devuelve `true`, el hecho de llamar al método `close()` no tiene ningún efecto y el efecto de llamar a cualquier otro método es indefinido.

Conceptos relacionados:

- "Interoperatividad entre JDBC y SQLj" en la página 284
- "Sesiones compartidas entre JDBC y SQLj" en la página 285

Archivos fuente y de salida para Java

Los archivos fuente tienen las siguientes extensiones:

.java Archivos fuente Java, que no necesitan precompilación. Puede compilar estos archivos con el compilador Java `javac` que se incluye con el entorno de desarrollo de Java.

.sqlj Archivos fuente SQLj, que necesitan conversión con el conversor `sqlj`. El conversor crea:

- Uno o más archivos de códigos de bytes `.class`
- Un archivo de perfil `.ser` por contexto de conexión

Los archivos de salida correspondientes tienen las siguientes extensiones:

.class Archivos compilados de códigos de bytes JDBC y SQLj.

.ser Archivos de perfil SQLj colocados en serie. El usuario crea paquetes en la base de datos para cada archivo de perfil con el programa de utilidad `db2profc`.

Bibliotecas de clases Java

DB2 Universal Database proporciona bibliotecas de clases para el soporte de JDBC y SQLj, que el usuario debe proporcionar en CLASSPATH o incluir en los applets del siguiente modo:

db2jcc.jar
Proporciona el controlador JDBC Tipo 4.

db2java.zip
Proporciona el controlador JDBC y clases de soporte de JDBC y SQLj, incluido soporte para procedimientos almacenados y UDF.

sqlj.zip
Proporciona los archivos de clases del conversor SQLj.

runtime.zip
Proporciona soporte en tiempo de ejecución de Java para aplicaciones y applets SQLj.

Dónde colocar clases Java

Puede utilizar archivos de clases Java individuales para procedimientos almacenados y UDF o recopilar los archivos de clases en archivos JAR e instalar el archivo JAR en la base de datos. Si decide utilizar archivos JAR, consulte la descripción sobre cómo registrar funciones y procedimientos almacenados Java para obtener más información.

Nota: si actualiza o sustituye archivos de clases de rutinas Java, debe emitir una sentencia CALL SQLJ.REFRESH_CLASSES() para permitir que DB2 cargue las clases actualizadas. Para obtener más información sobre la sentencia CALL SQLJ.REFRESH_CLASSES(), consulte la descripción sobre cómo actualizar clases Java para rutinas.

Para permitir que DB2 encuentre y utilice los procedimientos almacenados y las UDF, debe almacenar los archivos de clases correspondientes en el *directorio de funciones*, que es un directorio definido para el sistema operativo del siguiente modo:

Sistemas operativos Unix
`sqllib/function`

Sistemas operativos Windows
`nombre_instancia``\function`, donde *nombre_instancia* representa el valor del valor de registro específico de la instancia *DB2INSTPROF*.

Por ejemplo, el directorio de funciones correspondiente a un servidor Windows NT con DB2 instalado en el directorio `C:\sqllib` y sin valor de registro *DB2INSTPROF* especificado es:

```
C:\sqllib\function
```

Si decide utilizar archivos de clases individuales, debe almacenar los archivos de clases en el directorio adecuado para su sistema operativo. Si declara una clase de modo que forme parte de un paquete Java, cree los subdirectorios correspondientes en el directorio de funciones y coloque los archivos en el subdirectorio correspondiente. Por ejemplo, si crea una clase `ibm.tests.test1` para un sistema Linux, almacene el archivo de códigos de bytes de Java correspondiente (denominado test1.class) en sqllib/function/ibm/tests.

El JVM que DB2 invoca utiliza la variable de entorno CLASSPATH para localizar archivos Java. DB2 añade el directorio de funciones y sqllib/java/db2java.zip frente al valor CLASSPATH.

Para establecer el entorno de modo que la JVM pueda encontrar los archivos de clases Java, es posible que tenga que definir el parámetro de configuración *jdk_path* o utilizar el valor por omisión. Además, es posible que tenga que definir el parámetro de configuración *java_heap_sz* para aumentar el tamaño del almacenamiento dinámico correspondiente a la aplicación.

Tareas relacionadas:

- "Actualización de clases Java para tiempo de ejecución" en la página 289

Consulta relacionada:

- "Maximum Java Interpreter Heap Size configuration parameter - java_heap_sz" en el manual *Administration Guide: Performance*
- "Java Development Kit Installation Path configuration parameter - jdk_path" en el manual *Administration Guide: Performance*

Actualización de clases Java para tiempo de ejecución

Procedimiento:

Cuando actualice clases de rutinas Java, debe también emitir una sentencia CALL SQLJ.REFRESH_CLASSES() para hacer que DB2 cargue las nuevas clases. Si no emite la sentencia CALL SQLJ.REFRESH_CLASSES() después de actualizar las clases de rutinas Java, DB2 continúa utilizando las versiones anteriores de las clases. La sentencia CALL SQLJ.REFRESH_CLASSES() sólo se aplica a rutinas FENCED. DB2 renueva las clases cuando se produce una operación COMMIT o ROLLBACK.

Nota: no puede actualizar rutinas NOT FENCED sin detener y volver a iniciar el gestor de bases de datos.

Paquetes de Java

Para utilizar las bibliotecas de clases que se incluyen con DB2 en sus propias aplicaciones, debe incluir las sentencias `import` *package* adecuadas al principio de los archivos fuente. Puede utilizar los siguientes paquetes en las aplicaciones Java:

java.sql.*
: La API JDBC que se incluye en JDK. Debe importar este paquete en cada programa JDBC y SQLj.

sqlj.runtime.*
: Soporte de SQLj que se incluye con cada cliente DB2. Debe importar este paquete en cada programa SQLj.

sqlj.runtime.ref.*
: Soporte de SQLj que se incluye con cada cliente DB2. Debe importar este paquete en cada programa SQLj.

Variables del lenguaje principal en Java

Los argumentos de sentencias de SQL incorporado se pasan mediante *variables del lenguaje principal*, que son variables del lenguaje principal que aparecen en la sentencia de SQL. Las variables del lenguaje principal pueden tener un máximo de tres partes:

- Un prefijo consistente en dos puntos, **:**
- Un identificador opcional de modalidad de parámetro: IN, OUT o INOUT
- Una variable del lenguaje principal Java que es un identificador de Java correspondiente al parámetro, variable o campo

La evaluación de un identificador de Java no tiene efectos secundarios en un programa Java, así que puede aparecer varias veces en el código Java generado para sustituir una cláusula SQLj.

La siguiente consulta contiene la variable del lenguaje principal, *:x*, que es la variable, campo o parámetro *x* de Java visible en el ámbito que contiene la consulta:

```
SELECT COL1, COL2 FROM TABLE1 WHERE :x > COL3
```

Todas las variables del lenguaje principal especificadas en SQL compuesto son por omisión variables del lenguaje principal de entrada. Tiene que especificar

el identificador de modalidad de parámetro OUT o INOUT antes de la variable del lenguaje principal para marcarla como variable del lenguaje principal de salida. Por ejemplo:

```
#sql {begin compound atomic static
     select count(*) into :OUT count1 from employee;
     end compound}
```

Tipos de datos SQL soportados en Java

La tabla siguiente muestra el equivalente en Java de cada tipo de datos SQL, según la especificación JDBC correspondiente a correlaciones de tipos de datos. El controlador JDBC convierte los datos que se intercambian entre la aplicación y la base de datos utilizando el siguiente esquema de correlación. Utilice estas correlaciones en sus aplicaciones Java y sus UDF y procedimientos PARAMETER STYLE JAVA.

Nota: no hay soporte de variables del lenguaje principal para el tipo de datos DATALINK en ninguno de los lenguajes de programación soportados por DB2.

Tabla 17. Tipos de datos SQL correlacionados con declaraciones Java

Tipo de columna SQL	Tipo de datos Java	Descripción del tipo de columna SQL
SMALLINT (500 ó 501)	short	Entero con signo de 16 bits
INTEGER (496 ó 497)	int	Entero con signo de 32 bits
BIGINT (492 ó 493)	long	Entero con signo de 64 bits
REAL (480 ó 481)	float	Coma flotante de precisión simple
DOUBLE (480 ó 481)	double	Coma flotante de precisión doble
DECIMAL(*p*,*s*)(484 ó 485)	java.math.BigDecimal	Decimal empaquetado
CHAR(*n*) (452 ó 453)	java.lang.String	Serie de caracteres de longitud fija con longitud *n*, donde *n* va de 1 a 254
CHAR(*n*) FOR BIT DATA	byte[]	Serie de caracteres de longitud fija con longitud *n*, donde *n* va de 1 a 254
VARCHAR(*n*) (448 ó 449)	java.lang.String	Serie de caracteres de longitud variable
VARCHAR(*n*) FOR BIT DATA	byte[]	Serie de caracteres de longitud variable
LONG VARCHAR (456 ó 457)	java.lang.String	Serie de caracteres de longitud variable larga

Tabla 17. Tipos de datos SQL correlacionados con declaraciones Java (continuación)

Tipo de columna SQL	Tipo de datos Java	Descripción del tipo de columna SQL
LONG VARCHAR FOR BIT DATA	byte[]	Serie de caracteres de longitud variable larga
BLOB(*n*) (404 ó 405)	java.sql.Blob	Serie binaria de longitud variable y objeto grande
CLOB(*n*) (408 ó 409)	java.sql.Clob	Serie de caracteres de longitud variable y objeto grande
DBCLOB(*n*) (412 ó 413)	java.sql.Clob	Serie de caracteres de doble byte de longitud variable y objeto grande
DATE (384 ó 385)	java.sql.Date	Serie de caracteres de 10 bytes
TIME (388 ó 389)	java.sql.Time	Serie de caracteres de 8 bytes
TIMESTAMP (392 ó 393)	java.sql.Timestamp	Serie de caracteres de 26 bytes

Componentes de habilitación de Java

La habilitación de Java de DB2 tiene tres componentes independientes:

- Soporte para aplicaciones y applets cliente escritos en Java mediante JDBC para acceder a DB2
- Soporte de precompilación y vinculación para aplicaciones y applets cliente escritos en Java mediante SQLj para acceder a DB2
- Soporte para UDF y procedimientos almacenados de Java en el servidor

Conceptos relacionados:

- "Programación en SQLj" en la página 302

Tareas relacionadas:

- "Codificación de aplicaciones y applets JDBC" en la página 294

Soporte de aplicaciones y applets

Las secciones siguientes describen el soporte de aplicaciones y applets que ofrecen los distintos controladores JDBC.

Soporte de aplicaciones en Java con el controlador de tipo 2

La siguiente figura muestra cómo funciona una aplicación JDBC de tipo 2 con DB2. Las llamadas a JDBC se convierten en llamadas a DB2 a través de métodos nativos de Java. JDBC solicita flujo del cliente DB2 al servidor DB2.

Las aplicaciones SQLj utilizan este soporte de JDBC y además necesitan las clases en tiempo de ejecución de SQLj para autentificar y ejecutar los paquetes de SQL vinculados a la base de datos en la fase de precompilación y vinculación.

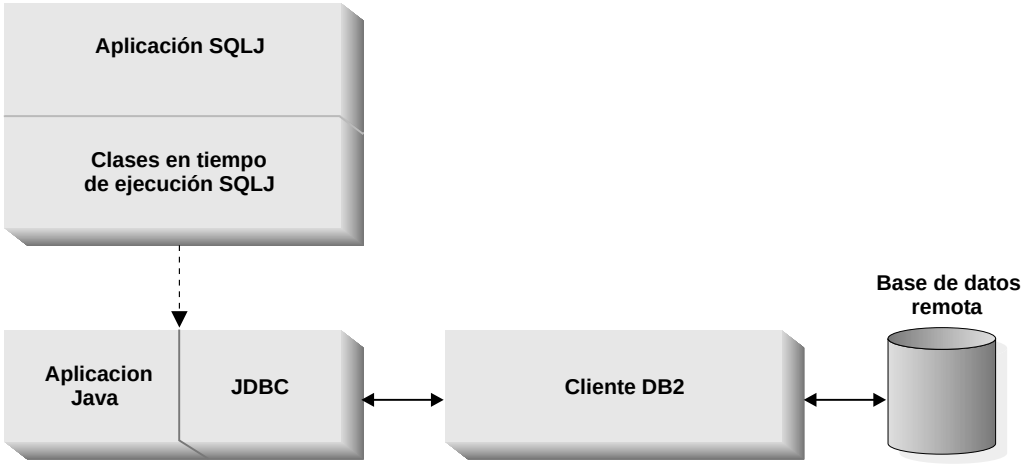


Figura 4. Soporte de aplicaciones con el controlador de tipo 2

Soporte de aplicaciones y applets en Java con el controlador de tipo 4

La siguiente figura muestra cómo funciona una aplicación o applet JDBC de DB2 de tipo 4. La aplicación o applet de tipo 4 se comunica directamente con la base de datos.

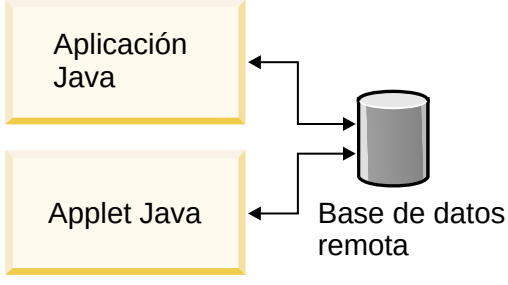


Figura 5. Soporte de aplicaciones y applets con el controlador de tipo 4

Soporte de applets en Java mediante el controlador de tipo 3

La siguiente figura muestra cómo funciona el *controlador de applets* de JDBC, también denominado *controlador de red*, con el controlador JDBC de tipo 3. El controlador de tipo 3 consiste en un cliente JDBC y un servidor JDBC, `db2jd`. El controlador del cliente JDBC se carga en el navegador Web junto con el

applet. Cuando el applet solicita una conexión con una base de datos de DB2, el cliente abre un zócalo TCP/IP para el servidor JDBC en la máquina en la que se está ejecutando el servidor Web. Una vez configurada una conexión, el cliente envía cada una de las siguientes peticiones de acceso a la base de datos desde el applet hasta el servidor JDBC a través de la conexión TCP/IP. Luego el servidor JDBC realiza las llamadas a la CLI (ODBC) correspondiente para realizar la tarea. Una vez finalizado el proceso, el servidor JDBC envía los resultados al cliente a través de la conexión.

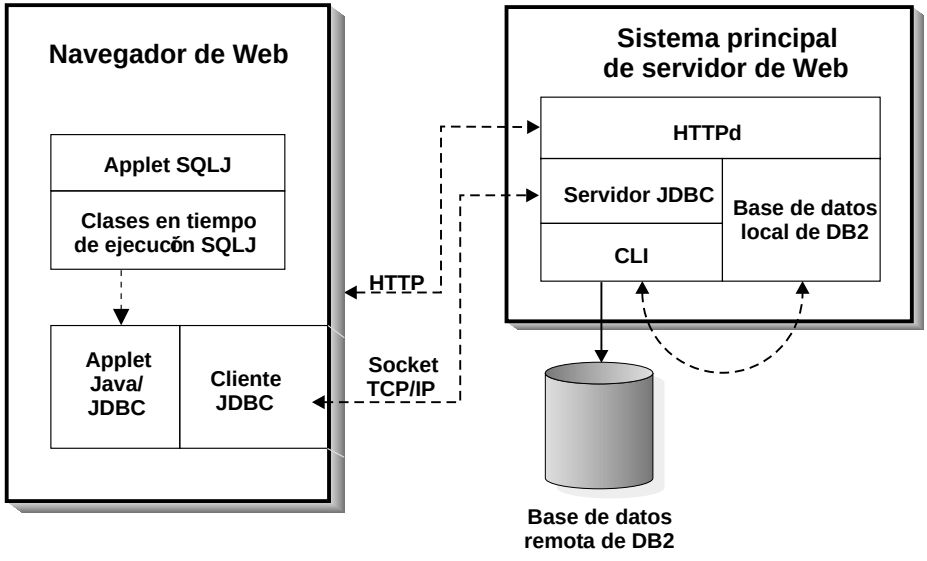


Figura 6. Implantación de applet Java de DB2 con el controlador JDBC de tipo 3

Los applets SQLj añaden el controlador del cliente SQLj en la parte superior del controlador del cliente JDBC, pero por lo demás funcionan igual que los applets JDBC.

Utilice el mandato `db2jstrt` para iniciar el servidor JDBC de DB2.

Programación en JDBC

Las secciones siguientes describen cómo crear aplicaciones JDBC.

Codificación de aplicaciones y applets JDBC

Las aplicaciones y applets JDBC suelen seguir una lógica de programa similar.

Procedimiento:

Cuando codifique una aplicación o applet JDBC, normalmente los codificará de modo que lleven a cabo las siguientes tareas:

1. Importar las clases y paquetes Java adecuados (`java.sql.*`).
2. Cargar el controlador JDBC adecuado:
   - Para JDBC de tipo 2, `COM.ibm.db2.jdbc.app.DB2Driver` para aplicaciones
   - Para JDBC de tipo 3, `COM.ibm.db2.jdbc.net.DB2Driver` para applets.

     Nota: se desaprueba el uso del controlador de tipo 3 en la Versión 8.
   - Para JDBC de tipo 4, `com.ibm.db2.jcc.DB2Driver` tanto para aplicaciones como para applets.
3. Conectar con la base de datos, especificando la ubicación con un URL según lo definido en la especificación JDBC y utilizando el subprotocolo `db2`.

   Los controladores de tipo 3 y 4 necesitan que proporcione el ID de usuario, contraseña, nombre de sistema principal y número de puerto. Para el controlador de tipo 3, el número de puerto es el número de puerto del servidor de applets. Para el controlador de tipo 4, el número de puerto es el número de puerto del servidor de DB2. El controlador de tipo 2 utiliza de forma implícita el valor por omisión para ID de usuario y contraseña del catálogo de clientes DB2, a no ser que especifique de forma explícita valores alternativos.
4. Pasar sentencias de SQL a la base de datos.
5. Recibir los resultados.
6. Cerrar la conexión.

Después de codificar el programa, compílelo igual que compilaría cualquier otro programa Java. No es necesario que realice ningún paso especial de precompilación ni de vinculación.

Conceptos relacionados:

• “Ejemplo de un programa JDBC” en la página 296

Especificación JDBC

Si la aplicación o el applet utiliza JDBC o SQLj, tiene que familiarizarse con la especificación JDBC, disponible de Sun Microsystems. Consulte el sitio Web de Java de DB2 para ver enlaces a recursos JDBC y SQLj. Esta especificación describe cómo llamar a las API de JDBC para acceder a una base de datos y manipular los datos de dicha base de datos.

Conceptos relacionados:

- “Componentes de habilitación de Java” en la página 292
- “JDBC 2.1” en la página 299

Ejemplo de un programa JDBC

Cada programa JDBC debe realizar los pasos siguientes:

1. Importar el paquete JDBC.

   Cada programa JDBC y SQLj debe importar el paquete JDBC.
2. Declarar un objeto Conexión.

   El objeto Conexión establece y gestiona la conexión con la base de datos.
3. Establecer la variable URL de la base de datos.

   El controlador de la aplicación DB2 acepta URL con el siguiente formato:

   `jdbc:db2:`*`nombre-basedatos`*
4. Conectar con la base de datos.

   El método `DriverManager.getConnection()` se suele utilizar con los siguientes parámetros:
   - `getConnection(String url)`, que establece una conexión con la base de datos especificada por un URL y utiliza el ID de usuario y contraseña por omisión.
   - `getConnection(String url, String userid, String password)`, que establece una conexión con la base de datos especificada por un URL y utiliza el ID de usuario y contraseña especificados mediante `userid` y `password` respectivamente.

   Nota: todos los programas de ejemplo de JDBC utilizan la clase `Db` definida en el programa **Util.java** para llevar a cabo la conexión. Puede volver a utilizar la clase `Db` del programa **Util.java** para conectar con una base de datos. Consulte los programas de ejemplo de JDBC para obtener más información.

El ejemplo de JDBC **TutMod.java** muestra algunas modificaciones básicas en la base de datos que incluyen sentencias INSERT, UPDATE y DELETE. Otro ejemplo de JDBC, **TbMod.java**, constituye un ejemplo más completo que muestra cómo insertar, actualizar y suprimir datos de una tabla. Este ejemplo muestra muchas formas posibles de modificar datos de una tabla.

Ejemplo de una sentencia INSERT

```
Statement stmt = con.createStatement();
stmt.executeUpdate(
  "INSERT INTO staff(id, name, dept, job, salary) " +
  "  VALUES (380, 'Pearce', 38, 'Clerk', 13217.50), "+
  "         (390, 'Hachey', 38, 'Mgr', 21270.00), " +
  "         (400, 'Wagland', 38, 'Clerk', 14575.00) ");
stmt.close();
```

Ejemplo de una sentencia UPDATE

```
Statement stmt = con.createStatement();
stmt.executeUpdate(
  "UPDATE staff " +
  "  SET salary = salary + 1000 " +
  "  WHERE id >= 310 AND dept = 84");
stmt.close();
```

Ejemplo de una sentencia DELETE

```
Statement stmt = con.createStatement();
stmt.executeUpdate("DELETE FROM staff " +
                   "  WHERE id >= 310 AND salary > 20000");
stmt.close();
```

Ejemplos relacionados:

- "TbMod.java -- How to modify table data (JDBC)"
- "TbMod.out -- HOW TO MODIFY TABLE DATA (JDBC)"
- "TutMod.java -- Modify data in a table (JDBC)"
- "TutMod.out -- HOW TO MODIFY DATA IN A TABLE (JDBC)"

Distribución de aplicaciones JDBC mediante el controlador de tipo 2

Distribuya sus aplicaciones JDBC como distribuiría cualquier otra aplicación Java. Como la aplicación utiliza el cliente DB2 para comunicarse con el servidor DB2, no debe tomar ninguna precaución especial sobre seguridad; la verificación de autorización la realiza el cliente DB2.

Para ejecutar la aplicación en una máquina cliente, debe instalar en dicha máquina:

- Una Java Virtual Machine (JVM), necesaria para ejecutar cualquier código Java
- Un cliente DB2, que también incluye el controlador JDBC de DB2

Para crear la aplicación, también debe instalar el JDK correspondiente a su sistema operativo.

Tareas relacionadas:

- "Creación de aplicaciones JDBC" en el manual *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

Distribución y ejecución de applets JDBC del controlador de tipo 4

Al igual que otros applets Java, el applet JDBC se distribuye por la red (intranet o Internet). Normalmente, incorporaría el applet en una página de lenguaje de marcación de hipertexto (HTML). Por ejemplo, para llamar al applet de ejemplo `DB2Applt.java`, (que se suministra en `sqllib/samples/java`), puede utilizar el siguiente código `<APPLET>`:

```
<applet code="DB2Applt.class" width=325 height=275 archive="db2jcc.jar">
   <param name="server" value="db_server">
   <param name="port" value="50006">
</applet>
```

Para ejecutar el applet, sólo necesita un navegador Web habilitado para Java en la máquina cliente (es decir, no necesita instalar un cliente DB2 en la máquina cliente). Cuando carga la página HTML, el código del applet indica al navegador que descargue el applet Java y la biblioteca de clases db2jcc.jar, que incluye el controlador JDBC de DB2 que implanta la clase `com.ibm.db2.jcc.DB2Driver`. Cuando el applet llama a la API JDBC para conectar con DB2, el controlador JDBC establece comunicaciones separadas con la base de datos DB2 a través del servidor de applets JDBC que se ejecuta en el servidor Web.

Nota: para asegurar que el navegador Web baja db2jcc.jar desde el servidor, asegúrese de que la variable de entorno CLASSPATH en el cliente *no* incluye db2jcc.jar. Es posible que el applet no funcione correctamente si el cliente utiliza una versión local de db2jcc.jar.

Tareas relacionadas:

- "Creación de applets JDBC" en el manual *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

Excepciones ocasionadas por una falta de correspondencia de archivos db2java.zip cuando se utiliza el controlador JDBC de tipo 3

Si utiliza el controlador JDBC de tipo 3, resulta esencial que el archivo db2java.zip que utiliza el applet Java esté al mismo nivel de FixPak que el servidor de applets JDBC. Bajo circunstancias normales, db2java.zip se descarga desde el servidor Web en el que se ejecuta el servidor de applets JDBC. Esto asegura una correspondencia. Sin embargo, si la configuración tiene el applet Java cargando db2java.zip desde otra ubicación, puede producirse una discrepancia. Antes de DB2 Versión 7.1 FixPak 2, esto podía dar lugar a errores inesperados. Desde DB2 Versión 7.1 FixPak 2, se hace cumplir de forma estricta la correspondencia de niveles de FixPak entre los dos archivos en el momento de la conexión. Si se detecta una discrepancia, se rechaza la conexión y el cliente recibe una de las excepciones siguientes:

- Si db2java.zip está al nivel de DB2 Versión 7.1 FixPak 2 o posterior:

  ```
  COM.ibm.db2.jdbc.DB2Exception: [IBM][Controlador JDBC]
  CLI0621E  Configuración de servidor JDBC no soportada.
  ```

- Si db2java.zip es anterior a DB2 Versión 7.1 FixPak 2:

  ```
  COM.ibm.db2.jdbc.DB2Exception: [IBM][Controlador JDBC]
  CLI0601E  Se ha cerrado la sentencia o descriptor de contexto de
            sentencia no válidos. SQLSTATE=S1000
  ```

Si se produce una discrepancia, el servidor de applets JDBC registra uno de los siguientes mensajes en el archivo jdbcerr.log:

- Si el servidor de applets JDBC está al nivel de DB2 Versión 7.1 FixPak o posterior:

```
jdbcFSQLConnect: Las versiones de servidor y cliente de applet
   JDBC (db2java.zip) no coinciden. La conexión no puede proseguir.
   einfo= -111
```

- Si el servidor de applets JDBC es anterior a DB2 Versión 7.1 FixPak 2:

```
jdbcServiceConnection(): Se ha recibido una petición no válida. einfo= 0
```

Nota: puesto que el controlador de tipo 4 no tiene ningún componente de servidor JDBC, no se puede producir el tipo de discrepancia descrito anteriormente. Se recomienda modificar las aplicaciones para que utilicen el controlador de tipo 4.

JDBC 2.1

JDBC Versión 2.1 de Sun tiene dos partes definidas: la **API central** y la **API de paquete opcional**. Para obtener información sobre la especificación JDBC, consulte el sitio Web de Java de DB2 Universal Database.

Conceptos relacionados:

- "Restricciones de la API central de JDBC 2.1 por parte del controlador JDBC de DB2 de tipo 2" en la página 299
- "Soporte de la API de paquete opcional de JDBC 2.1 por parte del controlador de JDBC de DB2 de tipo 2" en la página 300
- "Restricciones de la API central JDBC 2.1 impuestas por el controlador JDBC de DB2 de tipo 4" en la página 300
- "Soporte de la API Paquete opcional de JDBC 2.1 por parte del controlador JDBC de DB2 de tipo 4" en la página 302

Tareas relacionadas:

- "Configuración del entorno Java" en el manual *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

Restricciones de la API central de JDBC 2.1 por parte del controlador JDBC de DB2 de tipo 2

El controlador JDBC de DB2 de tipo 2 da soporte a la API central de JDBC 2.1, aunque no da soporte a todas las funciones definidas en la especificación. El controlador JDBC de DB2 *no* da soporte a las siguientes funciones:

- Conjuntos de resultados actualizables
- Nuevos tipos de SQL (Array, Ref, Java Object, Struct)
- Correlación personalizada de tipos de SQL

- Conjuntos de resultados sensibles desplazables (tipo de desplazamiento ResultSet.TYPE_SCROLL_SENSITIVE)

Conceptos relacionados:

- "Soporte de la API de paquete opcional de JDBC 2.1 por parte del controlador de JDBC de DB2 de tipo 2" en la página 300

Restricciones de la API central JDBC 2.1 impuestas por el controlador JDBC de DB2 de tipo 4

El controlador JDBC de DB2 de tipo 4 da soporte a la API central JDBC 2.1, aunque no da soporte a todas las funciones definidas en la especificación. El controlador JDBC de DB2 *no* da soporte a las siguientes funciones:

- Conjuntos de resultados actualizables
- Nuevos tipos de SQL (Array, Ref, Java Object, Struct)
- Correlación personalizada de tipos de SQL

Soporte de la API de paquete opcional de JDBC 2.1 por parte del controlador de JDBC de DB2 de tipo 2

El controlador de JDBC de DB2 de tipo 2 da soporte a las siguientes funciones de la API de paquete opcional de JDBC 2.1:

- Java Naming and Directory Interface (JNDI) para dar nombre a bases de datos

  DB2 proporciona el siguiente soporte para la interfaz Javing Naming and Directory Interface (JNDI) para dar nombre a bases de datos:

  javax.naming.Context

  Esta interfaz la implanta `COM.ibm.db2.jndi.DB2Context`, el cual maneja el almacenamiento y recuperación de objetos DataSource. Para dar soporte a las asociaciones permanentes de nombres fuente de datos lógicos con información de bases de datos físicas, como nombres de bases de datos, estas asociaciones se guardan en un archivo denominado .db2.jndi. Para una aplicación, el archivo reside (o se crea si no existe ninguno) en el directorio especificado por la variable de entorno USER.HOME. Para un applet, debe crear este archivo en el directorio raíz del servidor web para facilitar la operación de `lookup()`. Los applets no dan soporte a los métodos `bind()`, `rebind()`, `unbind()` ni `rename()` de esta clase. Sólo las aplicaciones pueden vincular objetos DataSource a JNDI.

  javax.sql.DataSource

  Esta interfaz la implanta `COM.ibm.db2.jdbc.DB2DataSource`. Puede guardar un objeto de esta clase en cualquier implantación de `javax.naming.Context`. Esta clase también utiliza el soporte de sondeo de conexión.

DB2DataSource da soporte a los siguientes métodos:
- public void setDatabaseName(String databaseName)
- public void setServerName(String serverName)
- public void setPortNumber(int portNumber)

javax.naming.InitialContextFactory

Esta interfaz la implanta COM.ibm.db2.jndi.DB2InitialContextFactory, el cual crea una instancia de DB2Context. Las aplicaciones definen automáticamente para la variable de entorno JAVA.NAMING.FACTORY.INITIAL el valor COM.ibm.db2.jndi.DB2InitialContextFactory. Para utilizar esta clase en un applet, llame a InitialContext() mediante la siguiente sintaxis:

```
Hashtable env = new Hashtable( 5 );
env.put( "java.naming.factory.initial",
         "COM.ibm.db2.jndi.DB2InitialContextFactory" );
Context ctx = new InitialContext( env );
```

- Sondeo de conexiones

  DB2ConnectionPoolDataSource y DB2PooledConnection proporcionan los enganches necesarios para que pueda implantar su propio módulo de sondeo de conexiones del siguiente modo:

  javax.sql.ConnectionPoolDataSource

  Esta interfaz la implanta COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource y es una fábrica de objetos COM.ibm.db2.jdbc.DB2PooledConnection.

  javax.sql.PooledConnection

  Esta interfaz la implanta COM.ibm.db2.jdbc.DB2PooledConnection.

- API de transacciones de Java (JTA)

  DB2 da soporte a las API de transacciones de Java (JTA) mediante el controlador JDBC de DB2 de tipo 2. DB2 no proporciona soporte de JTA con los controladores de JDBC de DB2 de tipo 3 y de tipo 4.

  javax.sql.XAConnection

  Esta interfaz la implanta COM.ibm.db2.jdbc.DB2XAConnection.

  javax.sql.XADataSource

  Esta interfaz la implanta COM.ibm.db2.jdbc.DB2XADataSource y es una fábrica de objetos COM.ibm.db2.jdbc.DB2PooledConnection.

  javax.transactions.xa.XAResource

  Esta interfaz la implanta COM.ibm.db2.jdbc.app.DBXAResource.

  javax.transactions.xa.Xid

  Esta interfaz la implanta COM.ibm.db2.jdbc.DB2Xid.

Conceptos relacionados:

- "Restricciones de la API central de JDBC 2.1 por parte del controlador JDBC de DB2 de tipo 2" en la página 299

Soporte de la API Paquete opcional de JDBC 2.1 por parte del controlador JDBC de DB2 de tipo 4

El controlador JDBC de DB2 de tipo 4 da soporte a las siguientes funciones de la API Paquete opcional de JDBC 2.1:

- Java Naming and Directory Interface (JNDI) para dar nombre a bases de datos

  DB2 proporciona el siguiente soporte para la interfaz Javing Naming and Directory Interface (JNDI) para dar nombre a bases de datos:

  javax.sql.DataSource

  Esta interfaz la implanta `com.ibm.db2.jcc.DB2SimpleDataSource`. Puede guardar un objeto de esta clase en cualquier implantación de `javax.naming.Context`.

  `DB2SimpleDataSource` da soporte a los siguientes métodos:
  - public void setDatabaseName(String databaseName)
  - public void setServerName(String serverName)
  - public void setPortNumber(int portNumber)
  - public void setDescription(String description)
  - public void setLoginTimeout(int seconds)
  - public void setLogWriter(java.io.PrintWriter logWriter)
  - public void setPassword(String password)
  - public void setUser(String user)
  - public void setDriverType(int driverType)

    Nota: el único tipo soportado actualmente es el 4; la aplicación debe definir este tipo de forma explícita tras la construcción de una nueva `DB2SimpleDataSource`.

Programación en SQLj

Las secciones siguientes describen cómo crear aplicaciones SQLj.

Programación en SQLj

El soporte de SQLj de DB2 se basa en el estándar ANSI de SQLj. Consulte el sitio Web de Java de DB2 para ver un puntero al sitio Web de ANSI y a otros recursos de SQLj. Las siguientes secciones contienen una visión general de la programación en SQLj e información específica del soporte de SQLj de DB2.

Los siguientes tipos de construcciones SQL pueden aparecer en programas SQLj:

- Consultas; por ejemplo, expresiones y sentencias SELECT

- Sentencias de cambio de datos (DML) de SQL; por ejemplo, INSERT, UPDATE, DELETE
- Sentencias de datos; por ejemplo, FETCH, SELECT..INTO
- Control de Transacciones; por ejemplo, COMMIT, ROLLBACK, etc.
- Lenguaje de definición de datos (DDL, también denominado Lenguaje de manipulación de esquemas); por ejemplo, CREATE, DROP, ALTER
- Llamadas a procedimientos almacenados; por ejemplo, CALL MYPROC(:x, :y, :z)
- Invocaciones de funciones; por ejemplo, VALUES(MYFUN(:x))

Conceptos relacionados:

- “Soporte de DB2 para SQLj” en la página 303
- “Restricciones de DB2 en SQLj” en la página 304

Soporte de DB2 para SQLj

El soporte de SQLj de DB2 se proporciona mediante DB2 Application Development Client. Junto con el soporte de JDBC que proporciona el cliente DB2, el soporte de SQLj de DB2 le permite crear, construir y ejecutar aplicaciones, applets, procedimientos almacenados y funciones definidas por el usuario (UDF) de SQL incorporado para Java. Estas aplicaciones pueden contener SQL estático y utilizar sentencias de SQL incorporado que se vinculan a una base de datos DB2.

El soporte de SQLj que proporciona DB2 Application Development Client incluye:

- El conversor SQLj, `sqlj`, que sustituye sentencias de SQL incorporado del programa SQLj por sentencias fuente Java y genera un perfil colocado en serie que contiene información sobre las operaciones SQL encontradas en el programa SQLj. El conversor SQLj utiliza el archivo `sqllib/java/sqlj.zip`.
- Las clases en tiempo de ejecución de SQLj, disponibles en `sqllib/java/runtime.zip`.
- El personalizador de perfiles de SQLj de DB2, `db2profc`, que precompila las sentencias de SQL almacenadas en el perfil generado y genera un paquete en la base de datos DB2.
- La impresora de perfiles de SQLj de DB2, `db2profp`, que imprime el contenido de un perfil personalizado de DB2 en texto plano.
- El instalador de auditores de perfiles de SQLj, `profdb`, que instala (o desinstala) auditores de clases de depuración en un conjunto existente de perfiles binarios. Una vez instalado, todas las llamadas a `RTStatement` y a `RTResultSet` realizadas durante el tiempo de ejecución de la aplicación se registran en un archivo (o salida estándar), que luego se puede consultar para verificar el comportamiento esperado y para ver si hay errores de

rastreo. Observe que sólo se auditan las llamadas realizadas a las interfaces de llamadas `RTStatement` y `RTResultSet` subyacentes durante el tiempo de ejecución.

- La herramienta de conversión de perfiles de SQLj, `profconv`, que convierte una instancia de perfiles colocados en secuencia a formato de código de bytes de clase. Algunos navegadores aún no dan soporte a la carga de un objeto colocado en serie desde un archivo de recursos asociado con el applet. Como solución temporal, tiene que ejecutar este programa de utilidad para realizar la conversión.

Para obtener más información sobre clases en tiempo de ejecución de SQLj, consulte el sitio Web de Java de DB2.

Conceptos relacionados:

- "Restricciones de DB2 en SQLj" en la página 304

Consulta relacionada:

- "db2profc - Mandato Personalizador de perfiles SQLj de DB2" en el manual *Consulta de mandatos*
- "db2profp - Mandato Impresora de perfiles SQLj de DB2" en el manual *Consulta de mandatos*

Restricciones de DB2 en SQLj

Cuando cree aplicaciones DB2 con SQLj, debe tener en cuenta las siguientes restricciones:

- El soporte de SQLj de DB2 cumple con las restricciones estándar de DB2 Universal Database sobre la emisión de sentencias de SQL.
- Una sentencia UPDATE y DELETE posicionada no es una subsentencia válida en una sentencia de SQL compuesto.
- La opción de precompilación ″DATETIME″ no recibe soporte. Sólo reciben soporte los formatos de fecha y hora de la Organización Internacional de Estándares (ISO).
- La opción de precompilación ″PACKAGE USING package-name″ especifica el nombre del paquete que debe generar el conversor. Si no se especifica ningún nombre, se utiliza el nombre del perfil (menos extensión y convertido a mayúsculas). La longitud máxima es de 8 caracteres. Puesto que el nombre del perfil de SQLj tiene el sufijo _SJProfile*N*, donde *N* es el número clave del perfil, el nombre del perfil siempre tendrá más de 8 caracteres. El nombre de paquete por omisión se construirá concatenando los primeros (8 - *pfKeyNumLen*) caracteres del número de perfil y el número clave del perfil, donde *pfKeyNumLen* es la longitud del número clave del

perfil en el nombre del perfil. Si la longitud del número clave del perfil es mayor que 7, se utilizarán los 7 últimos dígitos sin ningún aviso. Por ejemplo:

```
nombre del perfil                   nombre paquete por omisión
--------------------                ---------------------------
App_SJProfile1                      App_SJP1
App_SJProfile123                    App_S123
App_SJProfile1234567                A1234567
App_SJProfile12345678               A2345678
```

- Si se utiliza una variable del lenguaje principal `java.math.BigDecimal`, la precisión y escala de la variable del lenguaje principal no está disponible durante la conversión de la aplicación. Si la precisión y escala de la variable del lenguaje principal decimal no resultan obvias a partir del contexto de la sentencia en la que se utiliza, la precisión y escala se pueden especificar mediante CAST.
- No se puede utilizar una variable Java con el tipo `java.math.BigInteger` como variable del lenguaje principal en una sentencia de SQL.

Algunos navegadores aún no dan soporte a la carga de un objeto colocado en serie desde un archivo de recursos asociado con el applet. Obtendrá el siguiente mensaje de error si intenta cargar el applet Applt en dichos navegadores:

```
java.lang.ClassNotFoundException: Applt_SJProfile0
```

Como solución temporal, existe un programa de utilidad que convierte un perfil colocado en serie en un perfil almacenado en formato de clase Java. El programa de utilidad es una clase Java denominada `sqlj.runtime.profile.util.SerProfileToClass`. Toma un archivo de recursos de perfiles colocados en serie como entrada y genera una clase Java que contiene el perfil como salida. El perfil se puede convertir mediante el siguiente mandato:

```
profconv Applt_SJProfile0.ser
```

o

```
java sqlj.runtime.profile.util.SerProfileToClass Applt_SJProfile0.ser
```

Como resultado se crea la clase `Applt_SJProfile0.class`. Sustituya todos los perfiles en formato `.ser` que utiliza el applet por perfiles en formato `.class`.

Para un applet SQLj, necesita los archivos `db2java.zip` y `runtime.zip`. Si elige no empaquetar todas las clases del applet, las clases en `db2java.zip` y `runtime.zip` en un solo archivo Jar, coloque `db2java.zip` y `runtime.zip` (separados por una coma) en el parámetro archive en el código ″applet″. Para los navegadores que no dan soporte a varios archivos zip en el código

archive, especifique db2java.zip en el código archive y deszipee runtime.zip con las clases del applet en un directorio de trabajo al que pueda acceder el navegador web.

Sentencias de SQL incorporado en Java

Las sentencias de SQL estático en SQLj aparecen en *cláusulas SQLj*. Las cláusulas SQLj constituyen el mecanismo mediante el cual las sentencias de SQL de programas Java se comunican con la base de datos.

El conversor SQLj reconoce cláusulas SQLj y sentencias de SQL gracias a su estructura, del siguiente modo:

- las cláusulas SQLj empiezan con el símbolo #sql
- las cláusulas SQLj terminan con un punto y coma

Las cláusulas SQLj más sencillas son las *cláusulas ejecutables* y consisten en el símbolo #sql seguido de una sentencia de SQL entre llaves. Por ejemplo, la siguiente cláusula SQLj puede aparecer en cualquier lugar en el que pueda aparecer de forma válida una sentencia de Java. Su objetivo es eliminar todas las filas de la tabla denominada TAB:

```
#sql { DELETE FROM TAB };
```

En una cláusula ejecutable SQLj, los símbolos que aparecen dentro de las llaves son símbolos de SQL, excepto las variables del lenguaje principal. Todas las variables del lenguaje principal se distinguen por el carácter de dos puntos para que el conversor las pueda identificar. Los símbolos de SQL nunca aparecen fuera de las llaves de una cláusula ejecutable SQLj. Por ejemplo, el siguiente método Java inserta sus argumentos en una tabla de SQL. La parte principal del método cosiste en una cláusula ejecutable SQLj que contiene las variables del lenguaje principal *x*, *y* y *z*:

```
void m (int x, String y, float z) throws SQLException
{
    #sql { INSERT INTO TAB1 VALUES (:x, :y, :z) };
}
```

No inicialice sentencias de SQL estático en un bucle. Debe haber una sentencia física para cada sentencia de SQL estático. Por ejemplo, sustituya:

```
for( int i=0; i<2; i++){
   #sql [ctx] itr[i] = { SELECT id, name FROM staff };
}
```

por lo siguiente:

```
int i=0;
#sql [ctx] itr[i] = { SELECT id, name FROM staff };
i=1;
#sql [ctx] itr[i] = { SELECT id, name FROM staff };
```

En general, los símbolos de SQL no son sensibles a mayúsculas y minúsculas (excepto los identificadores delimitados por comillas dobles) y se pueden escribir en mayúsculas, en minúsculas o en una combinación de ambas. Sin embargo, los símbolos de Java son sensibles a mayúsculas y minúsculas. Para que los ejemplos resulten más claros, los símbolos de SQL que no son sensibles a mayúsculas y minúsculas aparecen en mayúsculas y los símbolos de Java aparecen en minúsculas o en una combinación de mayúsculas y minúsculas. El valor `null` en minúsculas se utiliza para representar un valor nulo de Java y el valor `NULL` en mayúsculas se utiliza para representar un valor nulo de SQL.

Conceptos relacionados:

- "Declaraciones y comportamiento del iterador en SQLj" en la página 307

Declaraciones y comportamiento del iterador en SQLj

A diferencia de las sentencias de SQL que recuperan datos de una tabla, las aplicaciones que llevan a cabo operaciones UPDATE y DELETE posicionadas o que utilizan iteradores con los atributos `holdability` o `returnability` necesitan dos archivos fuente Java. Declare el iterador como `public` en un archivo fuente, agregando la cláusula `with` e `implements` según convenga.

Para definir el valor del atributo `holdability` o `returnability`, debe declarar el iterador mediante la cláusula `with` para el atributo correspondiente. El siguiente ejemplo define para el atributo `holdability` el valor `true` para el iterador `WithHoldCurs`:

```
#sql public iterator WithHoldCurs with (holdability=true) (String EmpNo);
```

Los iteradores que llevan a cabo actualizaciones posicionadas necesitan una cláusula `implements` que implementa la interfaz `sqlj.runtime.ForUpdate`. Por ejemplo, supongamos que declara un iterador DelByName como este en `file1.sqlj`:

```
#sql public iterator DelByName implements sqlj.runtime.ForUpdate(String EmpNo);
```

Luego puede utilizar el iterador convertido y compilado en otro archivo fuente. Para utilizar el iterador:

1. Declare una instancia de la clase de iterador generada
2. Asigne la sentencia SELECT correspondiente a la operación UPDATE o DELETE posicionada a la instancia del iterador
3. Ejecute las sentencias UPDATE o DELETE posicionadas mediante el iterador

Para utilizar DelByName para una sentencia DELETE posicionada en `file2.sqlj`, ejecute sentencias como las del siguiente ejemplo:

```
{
   DelByName deliter; // Declarar objeto de clase DelByName
   String enum;
1  #sql deliter = { SELECT EMPNO FROM EMP WHERE WORKDEPT='D11'};
   while (deliter.next())
   {
2     enum = deliter.EmpNo(); // Obtener valor de la tabla de resultados
3     #sql { DELETE WHERE CURRENT OF :deliter };
      // Suprimir la fila en la que está posicionado el cursor
   }
}
```

Notas:

1. Esta cláusula 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 *deliter*.
2. Esta sentencia coloca el iterador en la siguiente fila que hay que suprimir.
3. Esta cláusula SQLj lleva a cabo la operación DELETE posicionada.

Conceptos relacionados:

• "Ejemplo de iteradores en un programa SQLj" en la página 308

Ejemplo de iteradores en un programa SQLj

El ejemplo de SQLj, **TbRead.sqlj**, utiliza SQL estático para recuperar datos de la tabla DEPARTMENT de la base de datos **sample** de DB2. Este ejemplo muestra dos tipos de iteradores:

- Vinculación con nombre a columnas

  Este iterador declara tipos y nombres de datos de columna y devuelve los valores de las columnas según el nombre de columna. El siguiente ejemplo se demuestra mediante la función `selectUsingNamedBindingToColumns()` del ejemplo **TbRead.sqlj**:

```
#sql iterator Named_Iterator(String deptnumb, String deptname);
Named_Iterator namedIter = null;

// declarar un cursor
#sql namedIter = {SELECT deptno as deptnumb, deptname
                    FROM department
                    WHERE admrdept = 'A00'};

// recuperar los valores de las columnas según el nombre de columna
while (namedIter.next())
{
  System.out.println( namedIter.deptnumb() + ", "+ namedIter.deptname() );
}

// cerrar el cursor
namedIter.close();
```

- Vinculación posicional a columnas

Este iterador declara tipos de datos de columna y devuelve los valores de las columnas por posición de columna. El siguiente ejemplo se demuestra mediante la función `selectUsingPositionalBindingToColumns()` del ejemplo **TbRead.sqlj**:

```
#sql iterator Positioned_Iterator(String, String);
Positioned_Iterator posIter;
String deptnumb = "";
String deptname = "";

// declarar cursor
#sql posIter = {SELECT deptno as deptnumb, deptname
                  FROM department
                  WHERE admrdept = 'A00'};

// captar el cursor
#sql {FETCH :posIter INTO :deptnumb, :deptname};

while (!posIter.endFetch())
{
  System.out.println( deptnumb + ", " + deptname );
  #sql {FETCH :posIter INTO :deptnumb, :deptname};
}

// cerrar el cursor
posIter.close();
```

Ejemplos relacionados:

- "TbRead.out -- HOW TO READ TABLE DATA (SQLJ)"
- "TbRead.sqlj -- How to read table data (SQLj)"

Llamadas a rutinas en SQLj

Las bases de datos pueden contener *procedimientos*, *funciones definidas por el usuario* y *métodos definidos por el usuario*. Los procedimientos, las funciones definidas por el usuario y los métodos definidos por el usuario son objetos de esquema con nombre que se ejecutan en la base de datos. Una cláusula ejecutable SQLj que aparece en una sentencia de Java puede llamar a un procedimiento mediante una sentencia CALL como la siguiente:

```
#sql { CALL SOME_PROC(:INOUT myarg) };
```

Los procedimientos pueden tener parámetros IN, OUT o INOUT. En el caso anterior, el valor de la variable del lenguaje principal *myarg* se modifica tras la ejecución de esta cláusula. Una cláusula ejecutable SQLj puede llamar a una función mediante la construcción de SQL `VALUES`. Por ejemplo, supongamos que tenemos una función `F` que devuelve un entero. El siguiente ejemplo ilustra una llamada a dicha función que luego asigna su resultado a la variable local de Java *x*:

```
{
    int x;
    #sql x = { VALUES( F(34) ) };
}
```

Conceptos relacionados:

- "Referencias a funciones" en el manual *Guía de desarrollo de aplicaciones: Programación de aplicaciones de servidor*
- "Vías de acceso y nombres de rutina" en el manual *Guía de desarrollo de aplicaciones: Programación de aplicaciones de servidor*
- "Referencias a procedimientos almacenados" en el manual *Guía de desarrollo de aplicaciones: Programación de aplicaciones de servidor*

Tareas relacionadas:

- "Creación de rutinas SQLJ" en el manual *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Invocación de procedimientos almacenados" en el manual *Guía de desarrollo de aplicaciones: Programación de aplicaciones de servidor*
- "Invocación de rutinas" en el manual *Guía de desarrollo de aplicaciones: Programación de aplicaciones de servidor*
- "Invocación de UDF" en el manual *Guía de desarrollo de aplicaciones: Programación de aplicaciones de servidor*
- "Invocación de funciones de tabla definidas por el usuario" en el manual *Guía de desarrollo de aplicaciones: Programación de aplicaciones de servidor*

Consulta relacionada:

- "Programas de ejemplo SQLJ" en el manual *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

Ejemplo de compilación y ejecución de un programa SQLj

Supongamos que tiene un programa SQLj denominado `MyClass`. Para ejecutar este programa, haría lo siguiente:

1. Convertir el archivo fuente SQLj (SQL incorporado para Java), `MyClass.sqlj`, con el conversor SQLj para generar el archivo fuente Java, `Myclass.java`. El conversor también crea los perfiles `MyClass_SJProfile0.ser`, `MyClass_SJProfile1.ser`, ... (un perfil para cada contexto de conexión):

   ```
   sqlj MyClass.sqlj
   ```

   Si utiliza el conversor `sqlj` sin especificar un archivo `sqlj.properties`, el conversor utiliza los siguientes valores:

```
sqlj.url=jdbc:db2:sample
sqlj.driver=COM.ibm.db2.jdbc.app.DB2Driver
sqlj.online=sqlj.semantics.JdbcChecker
sqlj.offline=sqlj.semantics.OfflineChecker
```

Si especifica un archivo `sqlj.properties`, asegúrese de que están definidas las siguientes opciones:

```
sqlj.url=jdbc:db2:dbname
sqlj.driver=COM.ibm.db2.jdbc.app.DB2Driver
sqlj.online=sqlj.semantics.JdbcChecker
sqlj.offline=sqlj.semantics.OfflineChecker
```

donde *dbname* es el nombre de la base de datos. También puede especificar estas opciones en la línea de mandatos. Por ejemplo, para especificar la base de datos `mydata` al convertir `MyClass`, puede emitir el siguiente mandato:

```
sqlj -url=jdbc:db2:mydata MyClass.sqlj
```

Observe que el conversor SQLj compila automáticamente el código fuente convertido en archivos de clases, a no ser que desactive de forma explícita la opción de compilación con la cláusula `-compile=false`.

2. Instalar Personalizadores de SQLj de DB2 en los perfiles generados y crear los paquetes de DB2 en la base de datos de DB2 *dbname*:

```
db2profc -user=user-name -password=user-password -url=jdbc:db2:dbname
  -prepoptions="bindfile using MyClass0.bnd package using MyClass0"
  MyClass_SJProfile0.ser
db2profc -user=user-name -password=user-password -url=jdbc:db2:dbname
  -prepoptions="bindfile using MyClass1.bnd package using MyClass1"
  MyClass_SJProfile1.ser
...
```

3. Ejecutar el programa SQLj:

```
java MyClass
```

El conversor genera la sintaxis de SQL correspondiente a la base de datos para la que está personalizado el perfil de SQLj. Por ejemplo:

```
i = { VALUES (  F(:x) ) };
```

se convierte mediante el conversor SQLj y se almacena como:

```
? = VALUES (F (?))
```

en el perfil generado. Cuando establezca conexión con una base de datos DB2 Universal Database, DB2 personalizará la sentencia VALUE como:

```
VALUES(F(?)) INTO ?
```

pero cuando establezca conexión con una base de datos DB2 Universal Database para OS/390 y z/OS, DB2 personalizará la sentencia VALUE como:

```
SELECT F(?) INTO ? FROM SYSIBM.SYSDUMMY1
```

Si ejecuta el personalizador de perfiles de SQLj de DB2, db2profc, contra una base de datos DB2 Universal Database y genera un archivo de vinculación, no puede utilizar dicho archivo de vinculación para vincular con una base de datos DB2 para OS/390 si hay una cláusula VALUES en el archivo de vinculación. Esto también se aplica si se genera un archivo de vinculación contra una base de datos DB2 para OS/390 y si se intenta vincularla con una base de datos DB2 Universal Database.

Tareas relacionadas:

- "Creación de applets SQLJ" en el manual *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Creación de aplicaciones SQLJ" en el manual *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Creación de rutinas SQLJ" en el manual *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Creación de programas SQLJ" en el manual *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

Opciones del conversor SQLj

El conversor SQLj da soporte a las mismas opciones de precompilación que el mandato PRECOMPILE de DB2, con las siguientes excepciones:

```
CONNECT
DISCONNECT
DYNAMICRULES
NOLINEMACRO
OPTLEVEL
OUTPUT
SQLCA
SQLFLAG
SQLRULES
SYNCPOINT
TARGET
WCHARTYPE
```

Para imprimir el contenido de los perfiles que genera el conversor SQLj en texto plano, utilice el programa de utilidad profp del siguiente modo:

```
profp MyClass_SJProfile0.ser
profp MyClass_SJProfile1.ser
  ...
```

Para imprimir el contenido de la versión personalizada de DB2 del perfil en texto plano, utilice el programa de utilidad db2profp del siguiente modo, donde *dbname* es el nombre de la base de datos:

```
db2profp -user=user-name -password=user-password -url=jdbc:db2:dbname
         MyClass_SJProfile0.ser
db2profp -user=user-name -password=user-password -url=jdbc:db2:dbname
         MyClass_SJProfile1.ser
    ...
```

Resolución de problemas de aplicaciones Java

Las secciones siguientes describen los recursos de rastreo disponibles para Java y los valores de SQLSTATE y SQLCODE de Java.

Recursos de rastreo en Java

Tanto el recurso de rastreo de CLI/ODBC/JDBC como el recurso de rastreo de DB2, db2trc, se pueden utilizar para diagnosticar problemas relacionados con programas JDBC o SQLj.

Nota: el controlador JDBC de tipo 4 no utiliza el recurso de rastreo de CLI/ODBC/JDBC. El rastreo correspondiente al controlador JDBC de tipo 4 se habilita mediante el método setLogWriter() en la API javax.sql.DataSource.

También puede instalar la función de rastreo de llamadas en tiempo de ejecución en programas SQLj. El programa de utilidad trabaja con los perfiles asociados con un programa. Supongamos que un programa utiliza un perfil denominado App_SJProfile0. Para instalar el rastreo de llamadas en el programa, utilice el siguiente mandato:

```
profdb App_SJProfile0.ser
```

El programa de utilidad profdb utiliza Java Virtual Machine para ejecutar el método main() de clase sqlj.runtime.profile.util.AuditorInstaller. Para obtener más detalles sobre el uso y las opciones de la clase AuditorInstaller, visite el sitio Web de Java de DB2.

Conceptos relacionados:

- “Recurso de rastreo de CLI/ODBC/JDBC” en la página 313
- “Archivos de rastreo de CLI y JDBC” en la página 324

Consulta relacionada:

- “db2trc - Mandato Rastrear” en el manual *Consulta de mandatos*

Recurso de rastreo de CLI/ODBC/JDBC

Este tema trata los siguientes aspectos:

- “Configuración del rastreo CLI de DB2 y JDBC de DB2” en la página 314

- “Opciones de rastreo de CLI de DB2 y el archivo db2cli.ini” en la página 315
- “Opciones de rastreo de JDBC de DB2 y el archivo db2cli.ini” en la página 320
- “Rastreo del controlador de CLI de DB2 frente a rastreo del Gestor de controladores de ODBC” en la página 322
- “Rastreos del controlador de CLI de DB2, del controlador de JDBC de DB2 y de DB2” en la página 323
- “Rastreos de CLI de DB2 y de JDBC de DB2 y procedimientos almacenados de CLI o Java” en la página 323

Los controladores de CLI de DB2 y DB2 JDBC proporcionan completos recursos de rastreo. Por omisión, estos recursos están inhabilitados y no utilizan ningún recurso de cálculo adicional. Cuando están habilitados, los recursos de rastreo generan uno o más archivos de registro de texto siempre que una aplicación accede al controlador adecuado (CLI de DB2 o JDBC de DB2). Estos archivos de registro proporcionan información detallada sobre:

- el orden en que la aplicación ha llamado a las funciones CLI o JDBC
- el contenido de los parámetros de entrada y salida que se han pasado a las funciones CLI o JDBC y se han recibido de las mismas
- los códigos de retorno y los mensajes de error o de aviso generados por las funciones CLI o JDBC

El análisis de rastreo de CLI de DB2 y JDBC de DB2 puede ayudar a los programadores de aplicaciones de varias formas. En primer lugar, los errores sutiles de lógica del programa y de inicialización de parámetros suelen ser evidentes en los rastreos. En segundo lugar, los rastreo de CLI de DB2 y JDBC de DB2 pueden sugerir formas de ajustar una aplicación o las bases de datos a las que accede. Por ejemplo, si un rastreo de CLI de DB2 muestra una tabla que se consulta varias veces en un determinado grupo de atributos, se puede crear un índice correspondiente a dichos atributos en la tabla para mejorar el rendimiento de la aplicación. Finalmente, el análisis de los archivos de rastreo de CLI de DB2 y JDBC de DB2 pueden ayudar a los programadores de aplicaciones a comprender el comportamiento de una interfaz o aplicación de otro proveedor.

Configuración del rastreo CLI de DB2 y JDBC de DB2:

Los parámetros de configuración correspondientes a los recursos de rastreo CLI de DB2 y JDBC de DB2 se leen del archivo de configuración de CLI de DB2 db2cli.ini. Por omisión, este archivo se encuentra en la vía de acceso \sqllib en la plataforma Windows y en la vía de acceso /sqllib/cfg en plataformas UNIX. Puede alterar temporalmente la vía de acceso por omisión estableciendo la variable de entorno DB2CLIINIPATH. En la plataforma

Windows, es posible que se encuentre un archivo db2cli.ini adicional en el directorio del perfil (o inicial) del usuario si hay fuentes de datos definidas por el usuario que se han definido mediante el Gestor de controladores de ODBC. Este archivo db2cli.ini alterará temporalmente el archivo por omisión.

Para ver los parámetros de configuración de rastreo actuales de db2cli.ini desde el procesador de línea de mandatos, emita el siguiente mandato:

```
db2 GET CLI CFG FOR SECTION COMMON
```

Hay tres formas de modificar el archivo db2cli.ini para configurar los recursos de rastreo de CLI de DB2 y de JDBC de DB2:

- utilizar el Asistente de configuración de DB2, si está disponible
- editar de forma manual el archivo db2cli.ini mediante un editor de texto
- emitir el mandato UPDATE CLI CFG desde el procesador de línea de mandatos

Por ejemplo, el siguiente mandato, emitido desde el procesador de línea de mandatos, actualiza el archivo db2cli.ini y habilita el recurso de rastreo JDBC:

```
db2 UPDATE CLI CFG FOR SECTION COMMON USING jdbctrace 1
```

Notas:

1. Normalmente, las opciones de configuración de rastreo CLI de DB2 y JDBC de DB2 sólo se leen desde el archivo de configuración db2cli.ini en el momento en que se inicializa una aplicación. Sin embargo, se puede utilizar una opción de rastreo especial de db2cli.ini, TRACEREFRESHINTERVAL, para indicar un intervalo en el que se deben volver a leer las opciones específicas de rastreo de CLI de DB2 desde el archivo db2cli.ini.
2. El recurso de rastreo de CLI de DB2 también se puede configurar de forma dinámica estableciendo los atributos de entorno SQL_ATTR_TRACE y SQL_ATTR_TRACEFILE. Estos valores alterarán temporalmente los valores contenidos en el archivo db2cli.ini.

Importante: Inhabilite los recursos de rastreo de CLI de DB2 y JDBC de DB2 cuando no los necesite. El rastreo innecesario puede reducir el rendimiento de la aplicación y puede generar archivos de registro de rastro no deseados. DB2 no suprime los archivos de rastreo generados y agrega la nueva información de rastreo a los archivos de rastreo existentes.

Opciones de rastreo de CLI de DB2 y el archivo db2cli.ini:

Cuando una aplicación que utiliza el controlador de CLI de DB2 se empieza a ejecutar, el controlador comprueba la existencia de opciones del recurso de rastreo en la sección [COMMON] del archivo db2cli.ini. Estas opciones de

rastreo son palabras clave de rastreo específicas que se establecen en determinados valores en el archivo db2cli.ini bajo la sección [COMMON].

Nota: puesto que las palabras clave de rastreo de CLI de DB2 aparecen en la sección [COMMON] del archivo db2cli.ini, sus valores se aplican a todas las conexiones de bases de datos a través del controlador de CLI de DB2.

Las palabras clave de rastreo de CLI de DB2 que se pueden definir son:

- TRACE
- TRACEFILENAME
- TRACEPATHNAME
- TRACEFLUSH
- TRACEREFRESHINTERVAL
- TRACECOMM
- TRACETIMESTAMP
- TRACEPIDTID
- TRACEPIDLIST
- TRACETIME
- TRACESTMTONLY

Nota: las palabras clave de rastreo de CLI de DB2 sólo se leen del archivo db2cli.ini una vez, en el momento de inicialización de la aplicación, a no ser que esté establecida la palabra clave TRACEREFRESHINTERVAL. Si esta palabra clave está establecida, las palabras clave TRACE y TRACEPIDLIST se vuelven a leer del archivo db2cli.ini en el intervalo especificado y se aplican, según proceda, a la aplicación que se esté ejecutando en ese momento.

TRACE = 0 | 1

La palabra clave TRACE determina si las palabras clave de rastreo de CLI de DB2 tienen o no efecto. Si esta palabra clave no está establecida o está establecida en el valor por omisión (0), el recurso de rastreo de CLI de DB2 está inhabilitado. Si esta palabra clave tiene el valor 1, el recurso de rastreo de CLI de DB2 está habilitado y se tienen en cuenta otras palabras clave de rastreo.

Por sí misma, la palabra clave TRACE tiene poco efecto (excepto para habilitar el proceso del recurso de rastreo de CLI de DB2). No se genera ninguna salida de rastreo a no ser que también se especifique la palabra clave TRACEPATHNAME o TRACEFILENAME.

TRACEFILENAME = <nombre_archivo_rastreo_completamente_calificado>

El nombre completamente calificado del archivo de registro en el que se graba toda la información de rastreo de CLI de DB2.

Si el archivo no existe, el recurso de rastreo de CLI de DB2 intentará crearlo. Si el archivo ya existe, la nueva información de rastreo de la sesión actual, si la hay, se agregará al contenido anterior de dicho archivo.

La opción de palabra clave TRACEFILENAME no se debe utilizar con aplicaciones de varios procesos o de varias hebras, puesto que la salida de rastreo correspondiente a todas las hebras o procesos se grabará en el mismo archivo de rastreo, y la salida correspondiente a cada hebra o proceso resultará difícil de descifrar. Además, se utilizan semáforos para controlar el acceso al archivo de rastreo compartido, lo que puede cambiar el comportamiento de las aplicaciones de varias hebras. No hay ningún nombre por omisión del archivo de registro de salida de rastreo de CLI de DB2.

TRACEPATHNAME = <nombre_vía_acceso_rastreo_completamente_calificado>

El nombre completamente calificado de la vía de acceso al directorio en el que se graba la información de rastreo de CLI de DB2. El recurso de rastreo de CLI de DB2 intentará generar un nuevo archivo de registro de rastreo cada vez que se ejecuta una aplicación que accede a la interfaz CLI de DB2. Si la aplicación tiene varias hebras, se generará un archivo de registro de rastreo independiente para cada hebra. Se utiliza de forma automática una concatenación del ID del proceso de la aplicación y el número de secuencia de la hebra para nombrar los archivos de registro de rastreo. No hay ninguna vía de acceso por omisión en la que se graban los archivos de registro de salida de rastreo de CLI de DB2, y la vía de acceso especificada debe existir en el momento en que se ejecuta la aplicación (el controlador de CLI de DB2 no creará la vía de acceso).

Nota: si se especifica tanto TRACEFILENAME como TRACEPATHNAME, la palabra clave TRACEFILENAME prevalece y TRACEPATHNAME se pasa por alto.

TRACEFLUSH = 0 | <cualquier entero positivo>

La palabra clave TRACEFLUSH especifica la frecuencia con la que la información de rastreo se graba en el archivo de registro de rastreo de CLI de DB2. Por omisión, TRACEFLUSH tiene el valor 0 y cada archivo de registro de rastreo de CLI de DB2 se mantiene abierto hasta que la aplicación o hebra de la que se realiza el rastreo termina de forma normal. Si la aplicación termina de forma anormal, es posible que se pierda la información de rastreo que no se grabó en el archivo de registro de rastreo.

Para asegurar la integridad y precisión de la información de rastreo que se graba en el archivo de registro de rastreo de CLI de DB2, se puede especificar la palabra clave TRACEFLUSH. Una vez se han

grabado n entradas de rastreo en el archivo de registro de rastreo, el controlador de CLI de DB2 cierra el archivo y lo vuelve a abrir, agregando las nuevas entradas de rastreo al final del archivo. Cada operación de cerrar y volver a abrir el archivo genera una actividad general de entrada/salida significativa y puede reducir considerablemente el rendimiento. *Cuando menor es el valor de la palabra clave TRACEFLUSH, mayor es el impacto del rastreo de CLI de DB2 en el rendimiento de la aplicación*.

El valor TRACEFLUSH=1 tiene un gran impacto sobre el rendimiento, pero asegura que cada entrada se graba en disco antes de que la aplicación continúe con la siguiente sentencia.

TRACEREFRESHINTERVAL = 0 | <cualquier entero positivo>
Si se establece TRACEREFRESHINTERVAL en un valor n entero positivo distinto del valor por omisión (0), el recurso de rastreo de CLI de DB2 vuelve a leer las palabras clave TRACE y TRACEPIDLIST del archivo db2cli.ini en el intervalo especificado (cada n segundos). Luego el recurso de rastreo de CLI de DB2 aplica estas palabras clave, según convenga, al rastreo que se está ejecutando en ese momento.

El resto de las palabras clave de configuración de rastreo de CLI de DB2 determinan qué información se graba en los archivos de registro de rastreo de CLI de DB2.

TRACECOMM = 0 | 1
Si se establece TRACECOMM en el valor por omisión (0), significa que no se incluirá información de comunicación cliente-servidor de DB2 en el rastreo de CLI de DB2. Si se establece TRACECOMM en 1, el rastreo de CLI de DB2 mostrará:

- qué funciones de CLI de DB2 se procesan por completo en el cliente y qué funciones de CLI de DB2 implican una comunicación con el servidor
- el número de bytes enviados y recibidos en cada comunicación con el servidor
- el tiempo empleado en la comunicación de datos entre el cliente y el servidor

TRACETIMESTAMP = 0 | 1 | 2 | 3
Si se establece TRACETIMESTAMP en un valor distinto del valor por omisión (0), significa que la indicación horaria actual o el tiempo de ejecución absoluto se añade al principio de cada línea de información de rastreo a medida que se graba en el archivo de registro de rastreo de CLI de DB2. Si se establece TRACETIMESTAMP en 1 se añade el tiempo de ejecución absoluto, en segundos y milisegundos, seguido de una indicación horaria. Si se establece TRACETIMESTAMP en 2 se

añade el tiempo de ejecución absoluto, en segundos y milisegundos. Si se establece TRACETIMESTAMP en 3 se añade la indicación horaria.

TRACEPIDTID = 0 | 1
Si se establece TRACEPIDTID en el valor por omisión (0), significa que la información de ID de hebra y proceso no se añadirá a cada línea del rastreo de CLI de DB2. Si se establece TRACEPIDTID en 1 significa que la información de ID de hebra y proceso se incluirá en el rastreo.

TRACEPIDLIST = <ningún valor> | <idp1,idp2, idp3,...>
Si se establece TRACEPIDLIST en su valor por omisión (ningún valor), o si no se establece, significa que el recurso de rastreo de CLI de DB2 realizará un rastreo de todos los procesos que acceden a la interfaz del controlador de CLI de DB2. Si se establece TRACEPIDLIST en una lista de uno o más valores de ID de proceso, delimitados por comas, se restringirán los rastreos de CLI generados a los procesos que aparecen en dicha lista.

TRACETIME = 0 | 1
Si se establece TRACETIME en su valor por omisión (1), o si no se establece, significa que el tiempo transcurrido entre las llamadas de funciones de CLI y la devolución de información se calculará e incluirá en el rastreo de CLI de DB2. Si se establece TRACETIME en 0 significa que el tiempo transcurrido entre llamadas de funciones de CLI y la devolución de la información no se calculará ni incluirá en el rastreo de CLI de DB2.

TRACESTMTONLY = 0 | 1
Si se establece TRACESTMTONLY en su valor por omisión (0), significa que todas las llamadas de funciones de CLI de DB2 se grabarán en el archivo de registro de rastreo de CLI de DB2. Si se establece TRACESTMTONLY en 1 significa que sólo al información relacionada con las llamadas de las funciones `SQLExecute()` y `SQLExecDirect()` se grabarán en el archivo de registro. Esta opción de rastreo puede resultar útil para determinar el número de veces que se ejecuta una sentencia en una aplicación.

A continuación se muestra un ejemplo de configuración de rastreo del archivo db2cli.ini en el que se utilizan estas palabras clave y valores de CLI de DB2:

```
[COMMON]
trace=1
TraceFileName=\temp\clitrace.txt
TRACEFLUSH=1
```

Notas:

1. Las palabras clave de rastreo de CLI NO son sensibles a mayúsculas y minúsculas. Sin embargo, puede que los valores de palabras claves

correspondientes a nombres de archivos y de vías de acceso sean sensibles a mayúsculas y minúsculas en algunos sistemas operativos (como en UNIX).

2. Si la palabra clave de rastreo de CLI de DB2 o su valor asociado en el archivo db2cli.ini no son válidos, el recurso de rastreo de CLI de DB2 los pasará por alto y utilizará el valor por omisión correspondiente a dicha palabra clave de rastreo.

Opciones de rastreo de JDBC de DB2 y el archivo db2cli.ini:

Cuando una aplicación que utiliza el controlador de JDBC de DB2 empieza a ejecutarse, el controlador también comprueba la existencia de opciones de recurso de rastreo en el archivo db2cli.ini. Al igual que sucede con las opciones de rastreo de CLI de DB2, las opciones de rastreo de JDBC de DB2 se especifican como pares palabra clave/valor situados bajo la sección [COMMON] del archivo db2cli.ini.

Nota: puesto que las palabras clave de rastreo de JDBC de DB2 aparecen en la sección [COMMON] del archivo db2cli.ini, sus valores se aplican a todas las conexiones de bases de datos a través del controlador de JDBC de DB2.

Las palabras clave de rastreo de JDBC de DB2 que se pueden definir son:

- JDBCTRACE
- JDBCTRACEPATHNAME
- JDBCTRACEFLUSH

JDBCTRACE = 0 | 1

La palabra clave JDBCTRACE controla si las otras palabras clave de rastreo de JDBC de DB2 tienen o no efecto en la ejecución del programa. Si se establece JDBCTRACE en su valor por omisión (0), se inhabilita el recurso de rastreo de JDBC de DB2; si se establece JDBCTRACE en 1, se habilita.

Por sí misma, la palabra clave JDBCTRACE tiene poco efecto y no genera ninguna salida de rastreo a no ser que también se especifique la palabra clave JDBCTRACEPATHNAME.

JDBCTRACEPATHNAME = <nombre_vía_acceso_rastreo_completamente_calificada>

El valor de JDBCTRACEPATHNAME es la vía de acceso completamente calificada al directorio en el que se graba toda la información de rastreo de JDBC de DB2. El recurso de rastreo de JDBC de DB2 intenta generar un nuevo archivo de registro de rastreo cada vez que se ejecuta una aplicación JDBC que utiliza el controlador de JDBC de DB2. Si la aplicación tiene varias hebras, se generará un

archivo de registro de rastreo independiente para cada hebra. Se utiliza de forma automática una concatenación del ID del proceso de aplicación, el número de secuencia de la hebra y una serie que identifica la hebra para nombrar los archivos de registro de rastreo. No hay ningún nombre de vía de acceso por omisión en el que se graben los archivos de registro de salida de rastreo de JDBC de DB2.

JDBCTRACEFLUSH = 0 | 1

La palabra clave JDBCTRACEFLUSH especifica la frecuencia con la que la información de rastreo se graba en el archivo de registro de rastreo de JDBC de DB2. Por omisión, JDBCTRACEFLUSH tiene el valor 0 y cada archivo de registro de rastreo de JDBC de DB2 se mantiene abierto hasta que la aplicación o hebra de la que se realiza el rastreo termina de forma normal. Si la aplicación termina de forma anormal, es posible que se pierda la información de rastreo que no se grabó en el archivo de registro de rastreo.

Para asegurar la integridad y precisión de la información de rastreo que se graba en el archivo de registro de rastreo de JDBC de DB2, la palabra clave JDBCTRACEFLUSH se puede establecer en 1. Después de que cada entrada de rastreo se haya grabado en el archivo de registro de rastreo, el controlador de JDBC de DB2 cierra el archivo y lo vuelve a abrir, añadiendo las nuevas entradas de rastreo al final del archivo. Esto garantiza que no se pierde información de rastreo.

Nota: *cada operación de cerrar y volver a abrir el archivo de registro de JDBC de DB2 genera una actividad general de entrada/salida significativa y puede reducir considerablemente el rendimiento de la aplicación*.

A continuación se muestra un ejemplo de configuración de rastreo del archivo db2cli.ini en el que se utilizan las palabras clave y valores de JDBC de DB2:

```
[COMMON]
jdbctrace=1
JdbcTracePathName=\temp\jdbctrace\
JDBCTRACEFLUSH=1
```

Notas:

1. Las palabras clave de rastreo de JDBC NO son sensibles a mayúsculas y minúsculas. Sin embargo, puede que los valores de palabras claves correspondientes a nombres de archivos y de vías de acceso sean sensibles a mayúsculas y minúsculas en algunos sistemas operativos (como en UNIX).
2. Si una palabra clave de rastreo de JDBC de DB2 o su valor asociado en el archivo db2cli.ini no son válidos, el recurso de rastreo de JDBC de DB2 los pasa por alto y utiliza el valor por omisión correspondiente a esa palabra clave.

3. Al habilitar el rastreo de JDBC de DB2 no se habilita el rastreo de CLI de DB2. Algunas versiones del controlador de JDBC de DB2 dependen del controlador de CLI de DB2 para acceder a la base de datos. Por lo tanto, puede que los programadores de Java también deseen habilitar el rastreo de CLI de DB2 para obtener información adicional sobre cómo las aplicaciones interactúan con la base de datos a través de las distintas capas de software. Las opciones de rastreo de JDBC de DB2 y de CLI de DB2 son independientes entre sí y se pueden especificar juntas en cualquier orden bajo la sección [COMMON] del archivo db2cli.ini.

Rastreo del controlador de CLI de DB2 frente a rastreo del Gestor de controladores de ODBC:

Es importante comprender las diferencias entre un rastreo del gestor de controladores de ODBC y un rastreo del controlador de CLI de DB2. Un rastreo del gestor de controladores de ODBC muestra las llamadas de funciones de ODBC realizadas por una aplicación ODBC a un gestor de controladores de ODBC. Por el contrario, un rastreo del controlador de CLI de DB2 muestra las llamadas de funciones realizadas por el gestor de controladores de ODBC al controlador de CLI de DB2 *en nombre de la aplicación*.

Un gestor de controladores de ODBC puede reenviar algunas llamadas de funciones directamente desde la aplicación al controlador de CLI de DB2. Sin embargo, el gestor de controladores de ODBC puede también retrasar o evitar el reenvío de algunas llamadas de funciones al controlador. El gestor de controladores de ODBC también puede modificar los argumentos de funciones de la aplicación o correlacionar funciones de la aplicación con otras funciones antes de reenviar la llamada al controlador de CLI de DB2.

Las razones para la intervención de llamadas de funciones de aplicación por parte del gestor de controladores de ODBC incluyen:

- Para las aplicaciones escritas con funciones de ODBC 2.0 que ya no se recomiendan en ODBC 3.0, las funciones antiguas se correlacionarán con funciones nuevas.
- Los argumentos de funciones de ODBC 2.0 que ya no se recomiendan en ODBC 3.0 se correlacionarán con argumentos equivalentes de ODBC 3.0.
- La biblioteca de cursores de Microsoft correlacionará llamadas como `SQLExtendedFetch()` con varias llamadas a `SQLFetch()` y a otras funciones soportadas para conseguir el mismo resultado final.
- La agrupación de conexiones del gestor de controladores de ODBC generalmente diferirá las peticiones `SQLDisconnect()` (o las evitará si la conexión se reutiliza).

Por esta y por otras razones, puede que los programadores de aplicaciones consideren el rastreo del gestor de controladores de ODBC un complemento útil al rastreo del controlador de CLI de DB2.

Para obtener más información sobre cómo capturar e interpretar rastreo del gestor de controladores de ODBC, consulte la documentación del gestor de controladores de ODBC. En las plataformas Windows, consulte el manual Microsoft ODBC 3.0 Software Development Kit and Programmer's Reference, también disponible en línea en: http://www.msdn.microsoft.com/.

Rastreos del controlador de CLI de DB2, del controlador de JDBC de DB2 y de DB2:

Internamente, algunas versiones del controlador de JDBC de DB2 utilizan el controlador de CLI de DB2 para el acceso a bases de datos. Por ejemplo, es posible que el controlador de JDBC de DB2 correlacione internamente el método getConnection() de Java con la función `SQLConnect()` de CLI de DB2. Como resultado, puede que los programadores de Java consideren el rastreo de CLI de DB2 un complemento útil al rastreo de JDBC de DB2.

El controlador de CLI de DB2 utiliza muchas funciones internas y específicas de DB2 para realizar su trabajo. Estas llamadas de funciones internas y específicas de DB2 se registran en el rastreo de DB2. Los programadores de aplicaciones no encontrarán útiles los rastreos de DB2, puesto que sólo están diseñados para ayudar al Servicio de IBM en la determinación y resolución de problemas.

Rastreos de CLI de DB2 y de JDBC de DB2 y procedimientos almacenados de CLI o Java:

En todas las plataformas de estación de trabajo, se pueden utilizar los recursos de rastreo de CLI de DB2 y de JDBC de DB2 para realizar el rastreo de procedimientos almacenados de CLI de DB2 y de JDBC de DB2.

La mayor parte de la información y las instrucciones sobre rastreo de CLI de DB2 y de JDBC de DB2 proporcionada en anteriores secciones es genérica y se aplica por igual a aplicaciones y a procedimientos almacenados. Sin embargo, a diferencia de las aplicaciones, que son clientes de un servidor de base de datos (y generalmente se ejecutan en una máquina distinta de la del servidor de base de datos), los procedimientos almacenados se ejecutan en el servidor de base de datos. Por lo tanto, debe seguir los siguientes pasos adicionales cuando efectúe un rastreo de procedimientos almacenados de CLI de DB2 o de JDBC de DB2:

- Asegúrese de que las opciones de palabras clave de rastreo están especificadas en el archivo db2cli.ini del servidor DB2.

- Si la palabra clave TRACEREFRESHINTERVAL no tiene un valor positivo distinto de cero, asegúrese de que todas las palabras clave estén correctamente configuradas antes del momento del arranque de la base de datos (es decir, cuando se emita el mandato db2start). Si se cambian los valores de rastreo mientras se está ejecutando el servidor de base de datos se pueden producir resultados inesperados. Por ejemplo, si se cambia el valor de TRACEPATHNAME mientras se está ejecutando el servidor, la próxima vez que se ejecute un procedimiento almacenado es posible que algunos archivos de rastreo se graben en la nueva vía de acceso mientras otros se graben en la vía de acceso original. Para asegurar la coherencia, vuelva a iniciar el servidor cada vez que se modifique una palabra clave de rastreo que no sea TRACE o TRACEPIDLIST.

Conceptos relacionados:

- "db2cli.ini Initialization File" en el manual *CLI Guide and Reference, Volume 1*
- "Archivos de rastreo de CLI y JDBC" en la página 324

Consulta relacionada:

- "SQLSetEnvAttr Function (CLI) - Set Environment Attribute" en el manual *CLI Guide and Reference, Volume 2*
- "db2trc - Mandato Rastrear" en el manual *Consulta de mandatos*
- "Mandato GET CLI CONFIGURATION" en el manual *Consulta de mandatos*
- "Mandato UPDATE CLI CONFIGURATION" en el manual *Consulta de mandatos*
- "Miscellaneous variables" en el manual *Administration Guide: Performance*
- "CLI/ODBC Configuration Keywords Listing by Category" en el manual *CLI Guide and Reference, Volume 1*

Archivos de rastreo de CLI y JDBC

Las aplicaciones que acceden a controladores de CLI de DB2 o JDBC de DB2 pueden utilizar los recursos de rastreo de CLI de DB2 o JDBC de DB2. Estos programas de utilidad registran todas las llamadas de funciones realizadas por los controladores de CLI de DB2 o JDBC de DB2 en un archivo de registro que resulta útil en la determinación de problemas. Este tema describe cómo acceder a estos archivos de registro que generan los recursos de rastreo y cómo interpretarlos:

- "Ubicación del archivo de rastreo de CLI y JDBC" en la página 325
- "Interpretación del archivo de rastreo de CLI" en la página 326
- "Interpretación del archivo de rastreo de JDBC" en la página 332

Ubicación del archivo de rastreo de CLI y JDBC:

Si se ha utilizado la palabra clave TRACEFILENAME en el archivo db2cli.ini para especificar un nombre de archivo completamente calificado, el archivo de registro de rastreo de CLI de DB2 estará en la ubicación especificada. Si se ha especificado un nombre de archivo relativo para el archivo de registro de rastreo de CLI de DB2, la ubicación de dicho archivo dependerá de lo que el sistema operativo considere que es la vía de acceso actual de la aplicación.

Nota: si el usuario que ejecuta la aplicación no tiene suficiente autoridad para grabar en el archivo de registro de rastreo de la vía de acceso especificada, no se generará ningún archivo ni se proporcionará ningún aviso ni error.

Si se ha utilizado la palabra clave TRACEPATHNAME o JDBCTRACEPATHNAME, o ambas, en el archivo db2cli.ini para especificar directorios completamente calificados, los archivos de registro de rastreo de CLI de DB2 y JDBC de DB2 estarán en la ubicación especificada. Si se ha especificado un nombre de directorio relativo para cualquiera de estos directorios de rastreo, o para ambos, el sistema operativo determinará su ubicación según lo que considere que es la vía de acceso actual de la aplicación.

Nota: si el usuario que ejecuta la aplicación no tiene suficiente autoridad para grabar archivos de rastreo en la vía de acceso especificada, no se generará ningún archivo ni se proporcionará ningún aviso ni error. Si la vía de acceso de rastreo especificada no existe, no se creará.

Los recursos de rastreo de CLI de DB2 y JDBC de DB2 utilizan de forma automática el ID de proceso y el número de secuencia de la hebra para nombrar los archivos de registro de rastreo cuando se han establecido las palabras clave TRACEPATHNAME y JDBCTRACEPATHNAME. Por ejemplo, un rastreo de CLI de DB2 de una aplicación con tres hebras puede generar los siguientes archivos de registro de rastreo de CLI de DB2: 100390.0, 100390.1, 100390.2.

De forma similar, un rastreo de JDBC de DB2 de una aplicación Java con dos hebras puede generar los siguientes archivos de registro de rastreo de JDBC: 7960main.trc, 7960Thread-1.trc.

Nota: Si el directorio de rastreo contiene archivos de registro de rastreo antiguos y nuevos, se puede utilizar la información de indicación horaria y fecha de los archivos para localizar los archivos de rastreo más recientes.

Si parece que no se ha creado ningún archivo de salida de rastreo de CLI de DB2 o JDBC de DB2:

- Compruebe que las palabras clave de configuración de rastreo estén correctamente establecidas en el archivo db2cli.ini. Una forma rápida de hacerlo consiste en emitir el mandato `db2 GET CLI CFG FOR SECTION COMMON` desde el procesador de línea de mandatos.
- Asegúrese de que la aplicación se vuelve a iniciar después de actualizar el archivo db2cli.ini. En concreto, los recursos de rastreo de CLI de DB2 y JDBC de DB2 se vuelven a inicializar durante el arranque de la aplicación. Una vez inicializado, el recurso de rastreo de JDBC de DB2 no se puede volver a configurar. El recurso de rastreo de CLI de DB2 se puede volver a configurar en el tiempo de ejecución, pero sólo si se ha especificado la palabra clave TRACEREFRESHINTERVAL correctamente antes del arranque de la aplicación.

  Nota: Sólo las palabras clave de CLI de DB2 TRACE y TRACEPIDLIST se pueden volver a ejecutar en el tiempo de ejecución. *Los cambios realizados en otras palabras clave de CLI de DB2, incluida TRACEREFRESHINTERVAL, no tienen efecto si no se vuelve a iniciar la aplicación*.
- Si la palabra clave TRACEREFRESHINTERVAL se ha especificado antes del arranque de la aplicación y la palabra clave TRACE se ha establecido inicialmente en 0, asegúrese de que ha transcurrido suficiente tiempo para que el recurso de rastreo de CLI de DB2 haya podido volver a leer el valor de la palabra clave TRACE.
- Si se utiliza la palabra clave TRACEPATHNAME o JDBCTRACEPATHNAME, o ambas, para especificar directorios de rastreo, asegúrese de que dichos directorios existen antes de iniciar la aplicación.
- Asegúrese de que la aplicación tiene acceso de grabación en el archivo de registro de rastreo o en el directorio de rastreo especificados.
- Compruebe la variable de entorno DB2CLIINIPATH. Si está establecida, los recursos de rastreo de CLI de DB2 y JDBC de DB2 esperan que el archivo db2cli.ini esté en la ubicación especificada por esta variable.
- Si la aplicación utiliza ODBC como interfaz con el controlador de CLI de DB2, compruebe que se ha llamado satisfactoriamente a una de estas funciones: `SQLConnect()`, `SQLDriverConnect()` o `SQLBrowseConnect()`. No se grabará ninguna entrada en los archivos de registro de rastreo de CLI de DB2 hasta que se haya establecido satisfactoriamente una conexión con la base de datos.

Interpretación del archivo de rastreo de CLI:

Los rastreos de CLI de DB2 siempre empiezan con una cabecera que identifica el ID de proceso y el ID de hebra de la aplicación que ha generado el rastreo,

la hora en que ha comenzado el rastreo e información específica del producto, como el nivel de build de DB2 y la versión del controlador de CLI de DB2. Por ejemplo:

```
1   [ Process: 1227, Thread: 1024 ]
2   [ Date, Time:           01-27-2002 13:46:07.535211 ]
3   [ Product:              QDB2/LINUX 7.1.0 ]
4   [ Level Identifier:     02010105 ]
5   [ CLI Driver Version:   07.01.0000 ]
6   [ Informational Tokens: "DB2 v7.1.0","n000510","" ]
```

Nota: en los ejemplos de rastreo que se muestran en esta sección se han añadido números de línea a la izquierda del rastreo. Estos números de línea se han añadido para clarificar la explicación y *no* aparecen en el rastreo de CLI de DB2 real.

Inmediatamente después de la cabecera del rastreo, suele haber varias entradas de rastreo relacionadas con el entorno y la asignación e inicialización del descriptor de contexto de la conexión. Por ejemplo:

```
7   SQLAllocEnv( phEnv=&bffff684 )
8       ---> Time elapsed - +9.200000E-004 seconds

9   SQLAllocEnv( phEnv=0:1 )
10      <--- SQL_SUCCESS   Time elapsed - +7.500000E-004 seconds

11  SQLAllocConnect( hEnv=0:1, phDbc=&bffff680 )
12      ---> Time elapsed - +2.334000E-003 seconds

13  SQLAllocConnect( phDbc=0:1 )
14      <--- SQL_SUCCESS   Time elapsed - +5.280000E-004 seconds

15  SQLSetConnectOption( hDbc=0:1, fOption=SQL_ATTR_AUTOCOMMIT, vParam=0 )
16      ---> Time elapsed - +2.301000E-003 seconds

17  SQLSetConnectOption( )
18      <--- SQL_SUCCESS   Time elapsed - +3.150000E-004 seconds

19  SQLConnect( hDbc=0:1, szDSN="SAMPLE", cbDSN=-3, szUID="", cbUID=-3,
                          szAuthStr="", cbAuthStr=-3 )
20      ---> Time elapsed - +7.000000E-005 seconds
21  ( DBMS NAME="DB2/LINUX", Version="07.01.0000", Fixpack="0x22010105" )

22  SQLConnect( )
23      <--- SQL_SUCCESS   Time elapsed - +5.209880E-001 seconds
24  ( DSN=""SAMPLE"" )

25  ( UID=" " )

26  ( PWD="*" )
```

En el ejemplo de rastreo anterior, observe que hay dos entradas para cada llamada de función de CLI de DB2 (por ejemplo, líneas 19-21 y 22-26 para la

llamada de función SQLConnect()). Esto siempre es así en los rastreos de CLI de DB2. La primera entrada muestra los valores de los parámetros de entrada que se han pasado a la llamada de función, mientras que la segunda entrada muestra los valores de parámetros de salida de función y el código de retorno que se ha devuelto a la aplicación.

El ejemplo de rastreo anterior muestra que la función SQLAllocEnv() ha asignado satisfactoriamente un descriptor de contexto de entorno (phEnv=0:1) en la línea 9. Este descriptor de contexto se ha pasado a la función SQLAllocConnect(), la cual ha asignado satisfactoriamente un descriptor de contexto de conexión de base de datos (phDbc=0:1) en la línea 13. Luego, se ha utilizado la función SQLSetConnectOption() para establecer el atributo SQL_ATTR_AUTOCOMMIT de la conexión de phDbc=0:1 en SQL_AUTOCOMMIT_OFF (vParam=0) en la línea 15. Finalmente, se ha llamado a la función SQLConnect() para conectar con la base de datos de destino (SAMPLE) en la línea 19.

En la entrada de rastreo de entrada de la función SQLConnect(), en la línea 21, se incluye el nivel de build y de FixPak del servidor de la base de datos de destino. Otra información que también puede aparecen en esta entrada de rastreo incluye palabras claves de la serie de conexión de entrada y las páginas de códigos del cliente y del servidor. Por ejemplo, supongamos que también apareciera la siguiente información en la entrada de rastreo de SQLConnect():

```
( Application Codepage=819, Database  Codepage=819,
  Char Send/Recv Codepage=819, Graphic Send/Recv Codepage=819,
  Application Char Codepage=819, Application Graphic Codepage=819 )
```

Esto significaría que la aplicación y el servidor de base de datos estarían utilizando la misma página de códigos (819).

La entrada de rastreo de retorno de la función SQLConnect() también contiene información de conexión importante (líneas 24-26 en el rastreo de ejemplo anterior). Información adicional que puede aparecer en la entrada de retorno incluye los valores de la palabra clave PATCH1 o PATCH2 que se aplican a la conexión. Por ejemplo, si se hubiera especificado PATCH2=27,28 en el archivo db2cli.ini bajo la sección COMMON, también aparecería la siguiente línea en la entrada de retorno de SQLConnect():

```
( PATCH2="27,28" )
```

Después de las entradas de rastreo relacionadas con el entorno y con la conexión se encuentran las entradas de rastreo relacionadas con sentencias. Por ejemplo:

```
27  SQLAllocStmt( hDbc=0:1, phStmt=&bffff684 )
28       ---> Time elapsed - +1.868000E-003 seconds
```

```
29  SQLAllocStmt( phStmt=1:1 )
30       <--- SQL_SUCCESS   Time elapsed - +6.890000E-004 seconds

31  SQLExecDirect( hStmt=1:1, pszSqlStr="CREATE TABLE GREETING (MSG
                                    VARCHAR(10))", cbSqlStr=-3 )
32       ---> Time elapsed - +2.863000E-003 seconds
33  ( StmtOut="CREATE TABLE GREETING (MSG VARCHAR(10))" )

34  SQLExecDirect( )
35       <--- SQL_SUCCESS   Time elapsed - +2.387800E-002 seconds
```

En el ejemplo de rastreo anterior, se ha utilizado el descriptor de contexto de la conexión de base de datos (phDbc=0:1) para asignar un descriptor de contexto de sentencia (phStmt=1:1) en la línea 29. Luego se ha ejecutado una sentencia de SQL no preparada en dicho descriptor de contexto de sentencia en la línea 31. Si se hubiera establecido la palabra clave TRACECOMM=1 en el archivo db2cli.ini, las entradas de rastreo de llamada de la función SQLExecDirect() mostrarían información adicional de comunicación cliente-servidor como la siguiente:

```
SQLExecDirect( hStmt=1:1, pszSqlStr="CREATE TABLE GREETING (MSG
                            VARCHAR(10))", cbSqlStr=-3 )
    ---> Time elapsed - +2.876000E-003 seconds
( StmtOut="CREATE TABLE GREETING (MSG VARCHAR(10))" )

    sqlccsend( ulBytes - 232 )
    sqlccsend( Handle - 1084869448 )
    sqlccsend( ) - rc - 0, time elapsed - +1.150000E-004
    sqlccrecv( )
    sqlccrecv( ulBytes - 163 ) - rc - 0, time elapsed - +2.243800E-002

SQLExecDirect( )
    <--- SQL_SUCCESS   Time elapsed - +2.384900E-002 seconds
```

Observe la información adicional de llamada de funciones sqlccsend() y sqlccrecv() de esta entrada de rastreo. La información de la llamada sqlccsend() revela la cantidad de datos que se han enviado del cliente al servidor, cuánto ha durado la transmisión y el éxito de dicha transmisión (0 = SQL_SUCCESS). Luego la información de la llamada sqlccrecv() revela el tiempo que el cliente ha esperado una respuesta del servidor y la cantidad de datos incluidos en la respuesta.

A menudo aparecerán varios descriptores de contexto de sentencias en el rastreo de CLI de DB2. Si presta atención al identificador del descriptor de contexto de la sentencia, puede seguir fácilmente el método de ejecución de un descriptor de contexto de sentencia de forma independiente de los demás descriptores de contexto de sentencia que aparecen en el rastreo.

Los métodos de ejecución de sentencias que aparecen en el rastreo de CLI de DB2 suelen ser más complicados que el del ejemplo anterior. Por ejemplo:

```
36  SQLAllocStmt( hDbc=0:1, phStmt=&bffff684 )
37      ---> Time elapsed - +1.532000E-003 seconds

38  SQLAllocStmt( phStmt=1:2 )
39      <--- SQL_SUCCESS   Time elapsed - +6.820000E-004 seconds

40  SQLPrepare( hStmt=1:2, pszSqlStr="INSERT INTO GREETING VALUES ( ? )",
                cbSqlStr=-3 )
41      ---> Time elapsed - +2.733000E-003 seconds
42  ( StmtOut="INSERT INTO GREETING VALUES ( ? )" )

43  SQLPrepare( )
44      <--- SQL_SUCCESS   Time elapsed - +9.150000E-004 seconds

45  SQLBindParameter( hStmt=1:2, iPar=1, fParamType=SQL_PARAM_INPUT,
                      fCType=SQL_C_CHAR, fSQLType=SQL_CHAR, cbColDef=14,
                      ibScale=0, rgbValue=&080eca70, cbValueMax=15,
                      pcbValue=&080eca4c )
46      ---> Time elapsed - +4.091000E-003 seconds

47  SQLBindParameter( )
48      <--- SQL_SUCCESS   Time elapsed - +6.780000E-004 seconds

49  SQLExecute( hStmt=1:2 )
50      ---> Time elapsed - +1.337000E-003 seconds
51  ( iPar=1, fCType=SQL_C_CHAR, rgbValue="Hello World!!!", pcbValue=14,
      piIndicatorPtr=14 )

52  SQLExecute( )
53      <--- SQL_ERROR   Time elapsed - +5.951000E-003 seconds
```

En el ejemplo de rastreo anterior, se ha utilizado el descriptor de contexto de conexión de base de datos (phDbc=0:1) para asignar un segundo descriptor de contexto de sentencia (phStmt=1:2) en la línea 38. Luego se ha preparado una sentencia de SQL con un marcador de parámetro en dicho descriptor de contexto de sentencia en la línea 40. Luego, se ha vinculado un parámetro de entrada (iPar=1) del tipo de SQL adecuado (SQL_CHAR) con el marcador de parámetro en la línea 45. Finalmente, se ha ejecutado la sentencia en la línea 49. Observe que tanto el contenido como la longitud del parámetro de entrada (rgbValue=″Hello World!!!″, pcbValue=14) se muestran en el rastreo en la línea 51.

La función SQLExecute() falla en la línea 52. Si la aplicación llama a una función de CLI de DB2 de diagnóstico, como SQLError(), para diagnosticar la causa del error, dicha causa aparecerá en el rastreo. Por ejemplo:

```
54  SQLError( hEnv=0:1, hDbc=0:1, hStmt=1:2, pszSqlState=&bffff680,
              pfNativeError=&bfffee78, pszErrorMsg=&bffff280,
              cbErrorMsgMax=1024, pcbErrorMsg=&bfffee76 )
55      ---> Time elapsed - +1.512000E-003 seconds

56  SQLError( pszSqlState="22001", pfNativeError=-302, pszErrorMsg="[IBM][CLI
       Driver][DB2/LINUX] SQL0302N  El valor de una variable del lenguaje
```

```
       principal de la sentencia EXECUTE u OPEN es demasiado alto para su
       uso correspondiente.
       SQLSTATE=22001", pcbErrorMsg=157 )
57      <--- SQL_SUCCESS   Time elapsed - +8.060000E-004 seconds
```

El mensaje de error que se devuelve en la línea 56 contiene el código de error nativo de DB2 que se ha generado (SQL0302N), el sqlstate correspondiente a dicho código (SQLSTATE=22001) y una breve descripción del error. En este ejemplo, el origen del error es evidente: en la línea 49, la aplicación intenta insertar una serie de 14 caracteres en una columna definida como VARCHAR(10) en la línea 31.

Si la aplicación no responde a un código de retorno de error o de aviso de una función de CLI de DB2 llamando a una función de diagnóstico como SQLError(), el mensaje de aviso o de error se tiene que grabar igualmente en el rastreo de CLI de DB2. Sin embargo, es posible que la ubicación de dicho mensaje en el rastreo no esté cerca de donde se ha producido realmente el error. Además, el rastreo indicará que la aplicación no ha recuperado el mensaje de error o de aviso. Por ejemplo, si no se recupera, es posible que el mensaje de error del ejemplo anterior no aparezca hasta más adelante y parezca que no esté relacionado con la llamada de función de CLI de DB2, como en el siguiente caso:

```
SQLDisconnect( hDbc=0:1 )
    ---> Time elapsed - +1.501000E-003 seconds
    sqlccsend( ulBytes - 72 )
    sqlccsend( Handle - 1084869448 )
    sqlccsend( ) - rc - 0, time elapsed - +1.080000E-004
    sqlccrecv( )
    sqlccrecv( ulBytes - 27 ) - rc - 0, time elapsed - +1.717950E-001
( Unretrieved error message="SQL0302N  El valor de una variable del lenguaje principal
  en la sentencia EXECUTE u OPEN es demasiado alto para su uso correspondiente.
  SQLSTATE=22001" )

SQLDisconnect( )
    <--- SQL_SUCCESS   Time elapsed - +1.734130E-001 seconds
```

La parte final de un rastreo de CLI de DB2 debe mostrar que la aplicación libera la conexión de base de datos y los descriptores de contexto de entorno que había asignado anteriormente en el rastreo. Por ejemplo:

```
58  SQLTransact( hEnv=0:1, hDbc=0:1, fType=SQL_ROLLBACK )
59      ---> Time elapsed - +6.085000E-003 seconds
60  ( ROLLBACK=0 )

61  SQLTransact( )
        <--- SQL_SUCCESS   Time elapsed - +2.220750E-001 seconds

62  SQLDisconnect( hDbc=0:1 )
63      ---> Time elapsed - +1.511000E-003 seconds

64  SQLDisconnect( )
```

```
65      <--- SQL_SUCCESS   Time elapsed - +1.531340E-001 seconds

66  SQLFreeConnect( hDbc=0:1 )
67      ---> Time elapsed - +2.389000E-003 seconds

68  SQLFreeConnect( )
69      <--- SQL_SUCCESS   Time elapsed - +3.140000E-004 seconds

70  SQLFreeEnv( hEnv=0:1 )
71      ---> Time elapsed - +1.129000E-003 seconds

72  SQLFreeEnv( )
73      <--- SQL_SUCCESS   Time elapsed - +2.870000E-004 seconds
```

Interpretación del archivo de rastreo de JDBC:

Los rastreos de JDBC de DB2 siempre comienzan con una cabecera que contiene información importante del sistema, como valores de variables clave de entorno, el nivel de JDK o JRE, el nivel del controlador de JDBC de DB2 y el nivel del build de DB2. Por ejemplo:

```
1   ===============================================================
2   |    Trace beginning on 2002-1-28 7:21:0.19
3   ===============================================================

4   System Properties:
5   ------------------
6   user.language = en
7   java.home = c:\Program Files\SQLLIB\java\jdk\bin\..
8   java.vendor.url.bug =
9   awt.toolkit = sun.awt.windows.WToolkit
10  file.encoding.pkg = sun.io
11  java.version = 1.1.8
12  file.separator = \
13  line.separator =
14  user.region = US
15  file.encoding = Cp1252
16  java.compiler = ibmjitc
17  java.vendor = IBM Corporation
18  user.timezone = EST
19  user.name = db2user
20  os.arch = x86
21  java.fullversion = JDK 1.1.8 IBM build n118p-19991124 (JIT ibmjitc
                       V3.5-IBMJDK1.1-19991124)
22  os.name = Windows NT
23  java.vendor.url = http://www.ibm.com/
24  user.dir = c:\Program Files\SQLLIB\samples\java
25  java.class.path =
         .:C:\Program Files\SQLLIB\lib;C:\Program Files\SQLLIB\java;
         C:\Program Files\SQLLIB\java\jdk\bin\
26  java.class.version = 45.3
27  os.version = 5.0
```

```
28  path.separator = ;
29  user.home = C:\home\db2user
30  ---------------------------------------
```

Nota: en los ejemplos de rastreo que se muestran en esta sección se han añadido números de línea a la izquierda del rastreo. Estos números de línea se han añadido para clarificar la explicación y *no* aparecen en el rastreo de JDBC de DB2 real.

Inmediatamente después de la cabecera de rastreo, suele haber varias entradas de rastreo relacionadas con la inicialización del entorno JDBC y el establecimiento de conexión de base de datos. Por ejemplo:

```
31  jdbc.app.DB2Driver -> DB2Driver() (2002-1-28 7:21:0.29)
32  | Loaded db2jdbc from java.library.path
33  jdbc.app.DB2Driver <- DB2Driver() [Time Elapsed = 0.01]

34  DB2Driver - connect(jdbc:db2:sample)

35  jdbc.app.DB2ConnectionTrace -> connect( sample, info, db2driver, 0, false )
                  (2002-1-28 7:21:0.59)
36  | 10: connectionHandle = 1
37  jdbc.app.DB2ConnectionTrace <- connect() [Time Elapsed = 0.16]

38  jdbc.app.DB2ConnectionTrace -> DB2Connection (2002-1-28 7:21:0.219)
39  | source = sample
40  | Connection handle = 1
41  jdbc.app.DB2ConnectionTrace <- DB2Connection
```

En el ejemplo de rastreo anterior, se ha efectuado una petición de carga del controlador de JDBC de DB2 en la línea 31. Esta petición ha resultado satisfactoria, tal como se notifica en la línea 33.

El recurso de rastreo de JDBC de DB2 utiliza clases Java específicas para capturar la información de rastreo. En el ejemplo de rastreo anterior, una de estas clases de rastreo, DB2ConnectionTrace, ha generado dos entradas de rastreo situadas en las líneas 35-37 y 38-41.

La línea 35 muestra que se invoca el método connect() y los parámetros de entrada de dicha llamada de método. La línea 37 muestra que la llamada al método connect() ha resultado satisfactoria, mientras que la línea 36 muestra el parámetro de salida de dicha llamada (Connection handle = 1).

Después de las entradas relacionadas con la conexión se suelen encontrar las entradas relacionadas con sentencias en el rastreo de JDBC. Por ejemplo:

```
42  jdbc.app.DB2ConnectionTrace -> createStatement() (2002-1-28 7:21:0.219)
43  | Connection handle = 1
44  | jdbc.app.DB2StatementTrace -> DB2Statement( con, 1003, 1007 )
          (2002-1-28 7:21:0.229)
45  | jdbc.app.DB2StatementTrace <- DB2Statement() [Time Elapsed = 0.0]
```

```
46  | jdbc.app.DB2StatementTrace -> DB2Statement (2002-1-28 7:21:0.229)
47  | | Statement handle = 1:1
48  | jdbc.app.DB2StatementTrace <- DB2Statement
49  jdbc.app.DB2ConnectionTrace <- createStatement - Time Elapsed = 0.01

50  jdbc.app.DB2StatementTrace -> executeQuery(SELECT * FROM EMPLOYEE WHERE
                                  empno = 000010) (2002-1-28 7:21:0.269)
51  | Statement handle = 1:1
52  | jdbc.app.DB2StatementTrace -> execute2( SELECT * FROM EMPLOYEE WHERE
                                    empno = 000010 ) (2002-1-28 7:21:0.269)
52  | | jdbc.DB2Exception -> DB2Exception() (2002-1-28 7:21:0.729)
53  | | | 10: SQLError = [IBM][CLI Driver][DB2/NT] SQL0401N  Los tipos de datos de
              los operandos de la operación "=" no son compatibles.
              SQLSTATE=42818
54  | | |     SQLState = 42818
55  | | |     SQLNativeCode = -401
56  | | |     LineNumber = 0
57  | | |     SQLerrmc = =
58  | | jdbc.DB2Exception <- DB2Exception() [Time Elapsed = 0.0]
59  | jdbc.app.DB2StatementTrace <- executeQuery - Time Elapsed = 0.0
```

En las líneas 42 y 43, la clase DB2ConnectionTrace ha notificado que se ha llamado al método de JDBC createStatement() con el descriptor de contexto de conexión 1. Dentro de dicho método, se ha llamado al método interno DB2Statement(), tal como indica otra clase del recurso de rastreo de JDBC de DB2, DB2StatementTrace. Observe que esta llamada al método interno aparece 'anidada' en la entrada de rastreo. Las líneas 47-49 muestran que los métodos han resultado satisfactorios y que se ha asignado el descriptor de contexto de sentencia 1:1.

En la línea 50, se ha realizado una llamada al método de consulta de SQL en la sentencia 1:1, pero la llamada ha generado una excepción en la línea 52. El mensaje de error se notifica en la línea 53 y contiene el código de error nativo de DB2 que se ha generado (SQL0401N), el sqlstate correspondiente a dicho código (SQLSTATE=42818) y una breve descripción del error. En este ejemplo, el error se debe a que la columna EMPLOYEE.EMPNO está definida como CHAR(6) y no como un valor entero, tal como se supone en la consulta.

Conceptos relacionados:

- "Recurso de rastreo de CLI/ODBC/JDBC" en la página 313

Consulta relacionada:

- "Miscellaneous variables" en el manual *Administration Guide: Performance*
- "TRACE CLI/ODBC Configuration Keyword" en el manual *CLI Guide and Reference, Volume 1*
- "TRACECOMM CLI/ODBC Configuration Keyword" en el manual *CLI Guide and Reference, Volume 1*

- "TRACEFILENAME CLI/ODBC Configuration Keyword" en el manual *CLI Guide and Reference, Volume 1*
- "TRACEPATHNAME CLI/ODBC Configuration Keyword" en el manual *CLI Guide and Reference, Volume 1*
- "TRACEPIDLIST CLI/ODBC Configuration Keyword" en el manual *CLI Guide and Reference, Volume 1*
- "TRACEREFRESHINTERVAL CLI/ODBC Configuration Keyword" en el manual *CLI Guide and Reference, Volume 1*

Valores de SQLSTATE y SQLCODE en Java

Si se produce un error de SQL, los programas JDBC y SQLj emiten una SQLException. Para recuperar los valores de SQLSTATE, SQLCODE o SQLMSG correspondientes a una instancia de una SQLException, invoque el método de la instancia correspondiente del siguiente modo:

Código de retorno de SQL	Método SQLException
SQLCODE	SQLException.getErrorCode()
SQLMSG	SQLException.getMessage()
SQLSTATE	SQLException.getSQLState()

Por ejemplo:

```
int sqlCode=0;      // Variable que va a contener SQLCODE
String sqlState="00000";  // Variable que va a contener SQLSTATE

try
{
  // Las sentencias de JDBC pueden emitir SQLExceptions
  stmt.executeQuery("Your JDBC statement here");

  // Las sentencias de SQLj también pueden emitir SQLExceptions
  #sql {..... your SQLj statement here ......};
}

/* Así es como puede comprobar la existencia de SQLCODEs y SQLSTATE */

catch (SQLException e)
{
  sqlCode = e.getErrorCode() // Obtener SQLCODE
  sqlState = e.getSQLState() // Obtener SQLSTATE

  if (sqlCode == -190 || sqlState.equals("42837"))
  {
    // Código para manejar SQLCODE -190 o SQLSTATE 42837
  }
  else
  {
    // Código para manejar otros errores
```

```
  }
  System.err.println(e.getMessage()) ; // Imprimir la excepción
  System.exit(1);         // Salir
}
```

Capítulo 11. Aplicaciones Java que utilizan WebSphere Application Servers

Servicios Web

La infraestructura de Internet está lista para dar soporte a la próxima generación de aplicaciones e-business, denominada servicios Web. Los servicios Web representan el siguiente nivel de función y eficacia en e-business. En concreto, los servicios Web son aplicaciones e-business mejoradas que resultan más fáciles de anunciar y de descubrir — por parte de otras empresas — porque se describen de forma más uniforme en Internet. Estas nuevas mejoras permiten conectar aplicaciones e-business con mayor facilidad tanto dentro como fuera de la empresa.

Un servicio Web es un conjunto de funciones de aplicación que realizan un servicio para un peticionario, como funciones informativas o transaccionales. Un servicio Web se puede describir y publicar en la red para que lo utilicen otros programas de la red. Ejemplos de servicios Web disponibles actualmente a nivel público incluyen un servicio de oferta de acciones, un servicio para obtener noticias de fuentes de noticias de la Web, un servicio para obtener mapas de eventos meteorológicos históricos por código postal, servicios de conversión de moneda y un servicio para obtener las condiciones de las autopistas en California. Puesto que los servicios Web son modulares, se pueden añadir servicios Web relacionados para formar un servicio Web de

mayor tamaño. Por ejemplo, podemos imaginar una aplicación inalámbrica compuesta de servicios Web independientes que obtiene ofertas de acciones, se suscribe a servicios de noticias, convierte moneda y gestiona agendas.

Los servicios Web amplían la audiencia de la Web para incluir programas y personas. En concreto, los servicios Web crean una arquitectura que permite que programas de software hagan lo que hacen las personas con la Web, es decir, acceder a documentos y ejecutar aplicaciones de forma general, sin necesitar un conocimiento ni software cliente específico de la aplicación. Una arquitectura que da soporte a los servicios Web proporciona la infraestructura para conseguir este objetivo.

La infraestructura de servicios Web se basa en XML (eXtensible Markup Language). Los mensajes y los datos fluyen entre un peticionario de servicios y un proveedor de servicios mediante XML. Las siguientes secciones describen brevemente los servicios Web, como se pueden transformar de forma dinámica datos DB2 a XML y el importante papel que juega DB2 en el mundo de los servicios Web.

Los servicios Web proporcionan un nivel de abstracción que facilita la tarea de acomodar una aplicación de empresa existente y convertirla en un servicio Web. Los servicios Web se basan en el formato de datos estándar XML y en mecanismos de intercambio de datos, que proporcionan tanto flexibilidad como independencia de la plataforma. Con servicios Web, los peticionarios no suelen conocer ni preocuparse de la implantación subyacente de servicios Web, lo que puede simplificar la integración de procesos empresariales heterogéneos.

Los servicios Web también le ofrecen una manera de hacer que sus procesos empresariales clave resulten accesibles para clientes, socios y proveedores. Por ejemplo, una compañía aérea puede proporcionar sus sistemas de reserva de vuelos como servicio Web para facilitar que sus clientes de empresas grandes integren el servicio en sus aplicaciones de planificación de viajes. Un proveedor puede hacer que sus niveles de inventario y sus precios resulten accesibles para sus principales compradores.

En un posible ejemplo, un comprador compara pedidos de compra entrantes con servicios de transporte:

1. El comprador accede a la base de datos local para seleccionar una lista de pedidos de compra.
2. Mientras consulta los detalles correspondientes a un determinado pedido de compra, el comprador selecciona un proveedor de servicios de transporte de una lista aprobada, una lista que se conserva en un registro privado.

3. Para cada proveedor, el comprador recibe ofertas de forma dinámica mediante las funciones de los servicios Web. Cada petición se vincula y se envía a la ubicación especificada en el registro, y el servicio Web del proveedor la procesa. El servicio Web del proveedor toma la entrada sobre la petición, accede a su base de datos y devuelve la oferta al peticionario.
4. Luego el comprador elige un proveedor de servicio basándose en los precios ofrecidos y la selección se añade a la página de pedidos de compra para reflejar la selección de un proveedor de servicio de envío.

Es probable que los servicios Web se extiendan donde las tecnologías existentes no lo han hecho. Los servicios Web aprovechan XML para la representación y el intercambio de datos y no necesitan complejas correlaciones que dependen del lenguaje ni vinculaciones en el momento de la compilación. Los servicios Web ofrecen facilidad tanto de desarrollo como de modificación. Además, los servicios Web no imponen estrechas relaciones síncronas entre peticionarios y proveedores de servicios. Esta característica facilita aún más la implantación de servicios Web en un entorno de Internet en el que no se puede controlar de forma estrecha el comportamiento de la red. La confianza en XML para el intercambio de datos y la abundancia de herramientas existentes y emergentes para la tecnología de servicios Web facilita bastante la implantación del primer servicio Web.

Arquitectura de servicios Web

La naturaleza de los servicios Web los convierte en componentes naturales de una arquitectura orientada a servicios. En una típica arquitectura orientada a servicios, los proveedores de servicios alojan un módulo de software accesible a través de la red, o servicio Web. Un proveedor de servicios define una descripción de servicio para un servicio Web y lo publica en un registro de servicios. Un peticionario de servicios utiliza una operación de búsqueda para recuperar la descripción del servicio del registro y utiliza la descripción del servicio para vincular con el proveedor de servicios e invocar o interactuar con la implantación del servicio Web.

En términos sencillos, un servicios Web se crea acomodando una aplicación de modo que se pueda acceder a la misma mediante mensajes XML estándares, que a su vez se acomodan de modo que enmascaran el protocolo de transporte subyacente. El servicio se puede poner a disponibilidad pública si se registra en un registro de formato estándar. Este registro permite a otras personas o aplicaciones encontrar y utilizar el servicio.

Las piezas de la arquitectura de servicios Web incluyen:

- Un servicio Web (un término general que se utiliza para describir software que se puede invocar sobre la Web)

- Mensajes específicos de cada aplicación que se envían en formatos de documento XML estándar que cumplen con la correspondiente descripción de servicio.
- Los mensajes XML están contenidos en sobres *Simple Object Access Protocol* (SOAP). SOAP es un protocolo de invocación de aplicaciones que define un protocolo sencillo para intercambiar información codificada como mensajes XML.

  Puesto que SOAP no realiza ningún supuesto sobre la implantación de puntos finales, el peticionario de servicios sólo tiene que crear una petición XML, enviarla a un proveedor de servicios y comprender la respuesta XML que se le devuelve. La implantación de DB2 se oculta al peticionario.

  Una petición SOAP consta del propio sobre, que contiene los espacios de nombres que utiliza el resto del mensaje SOAP, una cabecera opcional y la parte principal, que puede ser una llamada a un procedimiento remoto (RPC) o un documento XML.

  SOAP se basa en estándares existentes de Internet como HTTP y XML, pero se puede utilizar con cualquier protocolo de red, lenguaje de programación o modelo de codificación de datos. Por ejemplo, puede enviar mensajes SOAP sobre IBM MQSeries, FTP o incluso como mensajes de correo.
- La interfaz lógica y la implantación del servicio los describe el *Lenguaje de descripción de servicios Web* (WSDL). WSDL es un vocabulario XML que se utiliza para automatizar los detalles que intervienen en la comunicación entre las aplicaciones de servicios Web. Hay tres piezas de WSDL: un descriptor de tipo de datos (Esquema XML), una descripción de interfaz e información de vinculación. La descripción de interfaz se suele utilizar en el momento del desarrollo y la información de vinculación se puede utilizar en el momento del desarrollo o de la ejecución para invocar un determinado servicio en la ubicación especificada. La descripción del servicio resulta crucial para hacer que la arquitectura de servicios Web esté acoplada de forma holgada y para reducir la cantidad de conocimientos compartidos necesarios y programación personalizada entre proveedores de servicios y peticionarios de servicios.
- Para permitir que los peticionarios de servicios encuentren el servicio Web, puede publicar información descriptiva, como taxonomía, propiedad, nombre de empresa, tipo de empresa, etc., a través de un registro que cumpla con la especificación Uniform Description, Discovery and Integration (UDDI) o de algún otro registro XML. La información UDDI puede incluir un puntero a las interfaces WSDL, información de vinculación, así como el nombre real de la empresa (el nombre que permita a los usuarios comprender el objetivo del servicio Web). Un registro UDDI se puede buscar mediante programas, lo que permite a un peticionario de servicios vincular con un proveedor UDDI para encontrar más información sobre un servicio antes de utilizarlo realmente.

- La posibilidad de componer servicios Web juntos se suministra mediante *Web Services Flow Language* (WSFL). WSFL se puede utilizar para describir un proceso empresarial (es decir, un flujo de ejecución de principio a fin) o una descripción de las interacciones generales entre servicios Web variantes sin ninguna secuencia especificada.

Si se mira el modo en que estas especificaciones trabajan juntas, un servicio Web se puede definir como una aplicación modular que se puede:
- Describir mediante WSDL
- Publicar mediante UDDI
- Encontrar mediante UDDI
- Vincular mediante SOAP (o HTTP GET /POST)
- Invocar mediante SOAP (o HTTP GET/POST)
- Componer con otros servicios en nuevos servicios mediante WSFL

Puede restringir el acceso a servicios Web de forma parecida a como restringiría el acceso a sitios Web que no están disponibles para todo el mundo. WebSphere proporciona muchas opciones para controlar el acceso y para autentificación. La extensión de seguridad de SOAP, que se incluye con WebSphere Application Server 4.0, tiene como objetivo ser una arquitectura de seguridad basada en la especificación de seguridad de SOAP y en tecnologías de seguridad ampliamente aceptadas como Secure Socket Layer (SSL). Cuando se utiliza HTTP como mecanismo de transporte, hay distintas formas de combinar autentificación básica HTTP, SSL y signaturas SOAP para manejar requisitos variantes de seguridad y autentificación.

Acceso a datos

Las secciones siguientes describen cómo acceder a datos de DB2 con servicios Web.

Acceso a datos de DB2 a través de servicios Web

IBM está habilitando sus modelos de programación claves y servidores de aplicaciones con herramientas de desarrollo de Web para generar automáticamente servicios Web a partir de objetos y procedimientos almacenados existentes. Las secciones siguientes describen otra forma de someter consultas SQL y, si lo necesita, se ha planificado dar soporte al control del formato de las operaciones de los servicios Web:

- Almacenamiento o consulta basada en XML. Es decir, un documento XML se almacena en tablas relacionales DB2 y se compone de nuevo durante la recuperación. Este método de funcionamiento necesita la presencia de DB2 XML Extender.
- Operaciones basadas en SQL, como llamar a procedimientos almacenados o insertar, actualizar y suprimir datos de DB2.

Conceptos relacionados:

- “Acceso a datos de DB2 mediante consultas basadas en XML” en la página 342
- “Acceso a datos de DB2 mediante consultas basadas en SQL” en la página 342

Acceso a datos de DB2 mediante consultas basadas en XML

La consulta basada en XML le permite componer documentos XML a partir de datos relacionales. También puede dividir un documento XML en sus piezas de componentes y almacenarlo en tablas relacionales. Parte del soporte subyacente para esta función la ofrece DB2 XML Extender. Las operaciones de almacenamiento y recuperación se manejan mediante procedimientos almacenados especiales que se suministran con DB2 XML Extender.

Una de las entradas tanto en almacenamiento como en recuperación es el archivo de correlación especificado por el usuario que crea la asociación entre datos relacionales y la estructura del documento XML. Este archivo de correlación se denomina una *definición de acceso a documento* (DAD) y proporciona una forma de crear un documento XML con atributos y elementos XML denominados según necesite y con la forma que desee. En el centro de este enfoque está el movimiento y manipulación de documentos XML.

Acceso a datos de DB2 mediante consultas basadas en SQL

En servicios Web, consulta basada en SQL es sencillamente la posibilidad de enviar sentencias de SQL, incluidas llamadas a procedimientos almacenados, a DB2 y devolver resultados con la codificación por omisión. El centro de este enfoque consiste en mover datos a la base de datos y fuera de esta y no en formatear los resultados de una determinada manera.

La consulta basada en SQL no necesita el uso de DB2 XML Extender, porque no hay ninguna correlación definida por el usuario de datos SQL con atributos y elementos XML. En su lugar, los datos se devuelven utilizando únicamente una sencilla correlación de tipos de datos SQL, mediante nombres de columnas como elementos.

Sin embargo, si utiliza DB2 XML Extender para almacenar documentos XML dentro de una sola columna de una tabla, puede utilizar la consulta basada en SQL para recuperar estos documentos intactos como un objeto grande de caracteres (CLOB) o para invocar las funciones definidas por el usuario que extraen partes del documento. Otra función de DB2 XML Extender es la posibilidad de almacenar datos a los que se accede con frecuencia en tablas adicionales, lo que permite realizar búsquedas rápidas en documentos XML que están almacenados en columnas.

Otra cosa de utilidad que puede hacer con la consulta basada en SQL es invocar procedimientos almacenados de DB2. La naturaleza de los procedimientos almacenados permite convertirlos en servicios Web puesto que son en sí una encapsulación de lógica de programación y acceso a base de datos. Una invocación de servicio Web de un procedimiento almacenado permite proporcionar de forma dinámica parámetros de entrada y recuperar resultados.

Archivo de extensión de definición de acceso a documentos

Tanto los formatos de consulta basados en XML como los basados en SQL se controlan mediante un archivo de configuración denominado *extensión de definición de acceso a documentos* (DADX). El archivo de configuración DADX define las operaciones que puede llevar a cabo el servicio Web. Por ejemplo, puede tener un archivo DADX que especifique las operaciones para buscar todos los pedidos de piezas, buscar todos los pedidos de piezas con un determinado color y pedidos de piezas que sobrepasan un precio especificado. (El color o precio se puede especificar en el tiempo de ejecución como parámetros de entrada utilizando la notación de estilo de variable del lenguaje principal en la consulta.)

Después de crear un archivo DADX, WebSphere Studio Application Developer puede generar automáticamente la descripción WSDL de las interfaces y publicar las interfaces en un registro UDDI o en algún otro directorio de servicios. WebSphere Studio Application Developer también generará los objetos necesarios para desplegar el servicio Web en WebSphere Application Server y para generar los proxies del cliente, los cuales puede utilizar para pruebas y como base para la creación de la parte cliente de la aplicación Web.

Java 2 Platform Enterprise Edition

Las secciones siguientes describen Java 2 Platform Enterprise Edition (J2EE).

Visión general de Java 2 Platform Enterprise Edition (J2EE)

En el entorno empresarial global actual, las organizaciones tienen que ampliar su alcance, reducir sus costes y reducir sus tiempos de respuesta, proporcionando servicios a los que puedan acceder fácilmente sus clientes, empleados, proveedores y otros socios empresariales. Estos servicios deben tener las siguientes características:

- Altamente disponibles, para ajustarse a los requisitos del entorno empresarial global
- Seguros, para proteger la privacidad de los usuarios y la integridad de la empresa
- Fiables y escalables, de modo que las transacciones empresariales resulten precisas y se procesen con rapidez

En la mayoría de los casos, estos servicios se suministran con la ayuda de aplicaciones de varios enlaces en las que cada enlace tiene un objetivo específico. Java 2 Platform Enterprise Edition reduce el coste y la complejidad de desarrollar estos servicios de varios enlaces, lo que da lugar a servicios que se pueden desplegar rápidamente y se pueden mejorar fácilmente según los requisitos de la empresa.

Java 2 Enterprise Edition consigue estas ventajas definiendo una arquitectura estándar que se suministra como los siguientes elementos:

- Java 2 Enterprise Edition Application Model, un modelo de aplicación estándar para desarrollar servicios de clientes ligeros de varios enlaces
- Java 2 Enterprise Edition Platform, una plataforma estándar para alojar aplicaciones Java 2 Enterprise Edition
- Java 2 Enterprise Edition Compatibility Test Suite para verificar que un producto Java 2 Enterprise Edition Platform cumple con el estándar de Java 2 Enterprise Edition Platform
- Java 2 Enterprise Edition Reference Implementation para demostrar las funciones de Java 2 Enterprise Edition y para proporcionar una definición operativa de la plataforma Java 2 Enterprise Edition

Conceptos relacionados:

- "Java 2 Platform Enterprise Edition" en la página 344

Java 2 Platform Enterprise Edition

Java 2 Platform Enterprise Edition proporciona el entorno de tiempo de ejecución para alojar aplicaciones Java 2 Enterprise Edition. El entorno de tiempo de ejecución define cuatro tipos de componentes a los que un producto Java 2 Enterprise Edition debe dar soporte:

- Los clientes de aplicaciones son programas en lenguaje de programación Java que suelen ser programas GUI que se ejecutan en un sistema de escritorio. Los clientes de aplicaciones tienen acceso a todas las funciones del enlace medio de Java 2 Enterprise Edition.
- Los componentes applets y GUI que normalmente se ejecutan en un navegador web pero que se pueden ejecutar en otras aplicaciones o dispositivos que den soporte al modelo de programación de applets.
- Los servlets, JavaServer Pages (JSP), filtros y receptores de sucesos de la web que se suelen ejecutar en un navegador web y que pueden responder a peticiones HTTP procedentes de clientes web. Los servlets, JSP y filtros se pueden utilizar para generar páginas HTML que constituyen la interfaz de usuario de una aplicación. También se pueden utilizar para generar XML o datos en otro formato que consumen otros componentes de la aplicación. Los servlets, las páginas creadas con tecnología JSP, los filtros y los receptores de sucesos de la web reciben conjuntamente en esta

especificación el nombre *componentes de la web*. Las aplicaciones web constan de componentes de la web y de otros datos como páginas HTML.

- Los componentes Enterprise JavaBeans (EJB) se ejecutan en un entorno gestionado que da soporte a transacciones. Los Enterprise Beans suelen contener la lógica empresarial correspondiente a una aplicación Java 2 Enterprise Edition.

Los componentes de aplicaciones listados anteriormente se pueden dividir en tres categorías, según el modo en que se pueden desplegar y gestionar:

- Componentes que se despliegan, gestionan y ejecutan en un servidor Java 2 Enterprise Edition.
- Componentes que se despliegan y gestionan en un servidor Java 2 Enterprise Edition pero que se cargan en una máquina cliente y se ejecutan en la misma.
- Componentes cuyo despliegue y gestión no están completamente definidos por esta especificación. Los clientes de aplicaciones pueden encontrarse en esta categoría.

El soporte de tiempo de ejecución correspondiente a estos componentes se proporciona mediante *contenedores*.

Conceptos relacionados:

- "Contenedores de Java 2 Platform Enterprise Edition" en la página 345
- "Enterprise Java Beans" en la página 348

Contenedores de Java 2 Platform Enterprise Edition

Un contenedor proporciona una vista federada de las API subyacentes de Java 2 Platform Enterprise Edition a los componentes de la aplicación. Un producto Java 2 Platform Enterprise Edition típico proporcionará un contenedor para cada tipo de componente de la aplicación: contenedor de clientes de la aplicación, contenedor de applets, contenedor de web y contenedor de Enterprise Beans. Las herramientas de contenedor también comprenden los formatos de archivo para empaquetar los componentes de la aplicación para su despliegue.

La especificación necesita que estos contenedores proporcionen un entorno de tiempo de ejecución compatible con Java, según lo definido en la especificación J2SE de Java 2 Platform Enterprise Edition, Standard Edition V1.3.1. Esta especificación define un conjunto de servicios estándar a los que debe dar soporte cada producto Java 2 Enterprise Edition. Estos servicios estándar son:

- Servicio HTTP
- Servicio HTTPS
- API de transacciones Java

- Método de invocación remota
- IDL Java
- API JDBC
- Servicio de mensajes de Java
- Naming and Directory Interface de Java
- JavaMail
- Infraestructura de activación de JavaBeans
- API Java para el análisis XML
- Arquitectura de conectores
- Servicio de autentificación y autorización de Java

Conceptos relacionados:

- "Java Naming and Directory Interface (JNDI)" en la página 346
- "Enterprise Java Beans" en la página 348

Servidor Java 2 Platform Enterprise Edition

Como elemento subyacente de un contenedor Java 2 Platform Enterprise Edition se encuentra el servidor del que forma parte el contenedor. Un proveedor de productos Java 2 Enterprise Edition suele implantar las funciones de servidor Java 2 Platform Enterprise Edition mediante una infraestructura existente de proceso de transacciones en combinación con la tecnología J2SE. Las funciones del cliente Java 2 Platform Enterprise Edition suelen estar integradas en la tecnología J2SE.

Nota: IBM WebSphere Application Server es un servidor que cumple con las especificaciones de Java 2 Platform Enterprise Edition.

Requisitos de bases de datos de Java 2 Enterprise Edition

La plataforma Java 2 Enterprise Edition necesita una base de datos a la que se pueda acceder a través de la API JDBC para el almacenamiento de los datos de la empresa. Se puede acceder a la base de datos desde componentes de la web, Enterprise Beans y componentes cliente de la aplicación. No hace falta que se pueda acceder a la base de datos desde los applets.

Conceptos relacionados:

- "Consideraciones sobre la programación en Java" en la página 284

Java Naming and Directory Interface (JNDI)

JNDI permite que las aplicaciones basadas en la plataforma Java accedan a varios servicios de nomenclatura y directorio. Esto forma parte del conjunto de interfaces de programación de aplicaciones (API) de Java Enterprise. JNDI permite a los programadores crear aplicaciones portables habilitadas para distintos servicios de nomenclatura y directorio que incluyen sistemas de

archivos, servicios de directorio como Lightweight Directory Access Protocol (LDAP), Novell Directory Services y Network Information System (NIS) y sistemas de objetos distribuidos como Common Object Request Broker Architecture (CORBA), Invocación a métodos remotos (RMI) de Java y Enterprise JavaBeans (EJB).

La API JNDI tiene dos partes: una interfaz de nivel de aplicación que utilizan los componentes de la aplicación para acceder a los servicios de nomenclatura y directorio y una interfaz de proveedor de servicios para conectar con un proveedor de un servicio de nomenclatura y directorio.

Gestión de transacciones Java

Java 2 Enterprise Edition simplifica la programación de aplicaciones para la gestión de transacciones distribuidas. Java 2 Enterprise Edition incluye soporte de transacciones distribuidas a través de dos especificaciones, API de transacciones Java y Servicio de transacciones Java (JTS). JTA es una API de alto nivel, independiente de la implementación e independiente del protocolo, que permite a las aplicaciones y a los servidores de aplicaciones acceder a transacciones. Además, JTA está siempre habilitada.

JTA especifica interfaces Java estándar entre un gestor de transacciones y las partes que intervienen en un sistema de transacciones distribuidas: el gestor de recursos, el servidor de aplicaciones y las aplicaciones transaccionales.

JTS especifica la implantación de un Gestor de transacciones que da soporte a JTA e implanta la correlación Java de la especificación OMG Object Transaction Service (OTS) 1.1 al nivel que hay bajo la API. JTS propaga las transacciones mediante IIOP.

JTA y JTS permiten a los servidores de aplicaciones Java 2 Enterprise Edition evitar al desarrollador de componentes la tarea de gestión de transacciones. Los programadores pueden definir las propiedades transaccionales de la tecnología EJB basándose en componentes durante el diseño o despliegue mediante sentencias declarativas en el descriptor de despliegue. El servidor de aplicaciones se hace cargo de la responsabilidad de la gestión de transacciones.

En DB2 y en el entorno WebSphere Application Server, WebSphere Application Server asume la función de gestor de transacciones y DB2 actúa como un gestor de recursos. WebSphere Application Server implanta JTS y parte de JTA, y el controlador JDBC de DB2 también implanta parte de JTA de modo que WebSphere y DB2 pueden proporcionar transacciones distribuidas coordinadas.

Nota: no es necesario configurar DB2 de modo que esté habilitado para JTA en el entorno WebSphere porque el controlador JDBC de DB2 detecta automáticamente este entorno. Actualmente, sólo se proporciona soporte de JTA en el controlador JDBC de DB2 de tipo 2.

El controlador JDBC de DB2 proporciona dos clases DataSource:
- `COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource`
- `COM.ibm.db2.jdbc.DB2XADataSource`

WebSphere Application Server proporciona conexiones DB2 agrupadas con bases de datos. Si la aplicación va a participar en una transacción distribuida, se debe utilizar la clase `COM.ibm.db2.jdbc.DB2XADataSource` al definir fuentes de datos DB2 dentro de WebSphere Application Server.

Para ver información detallada sobre cómo configurar WebSphere Application Server con DB2, consulte WebSphere Application Server InfoCenter en:

`http://www-4.ibm.com/software/webservers/appserv/library.html`

Enterprise Java Beans

La arquitectura Enterprise Java Beans constituye la arquitectura de componentes para el desarrollo y despliegue de aplicaciones de empresa distribuidas basadas en componentes. Las aplicaciones escritas mediante la arquitectura de Enterprise Java Beans son escalables, transaccionales y seguras para múltiples usuarios. Estas aplicaciones se pueden escribir una vez y luego desplegar en cualquier plataforma de servidor que dé soporte a la especificación Enterprise Java Beans. Las aplicaciones Java 2 Enterprise Edition implantan componentes de empresa del servidor mediante Enterprise Java Beans (EJB) que incluyen beans de sesión y beans de entidad.

Los beans de sesión representan los servicios de la 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.

La aplicación de ejemplo EJB proporciona dos servicios de empresa. Un servicio permite al usuario acceder a información sobre un empleado (que se almacena en la tabla EMPLOYEE de la base de datos **sample**) mediante el número de empleado de dicho empleado. El otro servicio permite al usuario recuperar una lista de números de empleado de modo que el usuario pueda obtener un número de empleado para utilizarlo para consultar datos del empleado.

El siguiente ejemplo utiliza EJB para implantar una aplicación Java 2 Enterprise Edition para acceder a una base de datos DB2. El ejemplo utiliza la

arquitectura Modelo-Vista-Controlador (MVC). Se utiliza la JSP para implantar la Vista (el componente de presentación). Un servlet actúa como el controlador en el ejemplo. Controla el flujo de trabajo y delega la petición del usuario al Modelo que se implanta mediante Enterprise Java Beans. El componente Modelo del ejemplo consta de dos EJB, un bean de sesión y un bean de entidad. El bean de entidad CMP, Employee, representa los objetos transaccionales distribuidos que representan los datos permanentes de la tabla EMPLOYEE de la base de datos sample. El objetivo de utilizar el bean de permanencia gestionada por contenedor (CMP) es mostrar lo fácil que resulta utilizarlo en el desarrollo. El términos 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). Gracias a esta flexibilidad, aunque vuelva a desplegar el mismo bean de entidad en distintos servidores Java 2 Enterprise Edition que utilicen distintas bases de datos, el bean de sesión, AccessEmployee, actúa como fachada del bean de entidad y proporciona una estrategia uniforme de acceso de clientes. Este diseño de fachada reduce el tráfico en la red entre el cliente EJB y el bean de entidad y resulta más eficiente en transacciones distribuidas que cuando el cliente EJB accede al bean de entidad directamente. El acceso a la base de datos DB2 se puede proporcionar desde el bean de sesión o desde el bean de entidad. Los dos servicios de la aplicación de ejemplo demuestran ambos enfoques para acceder a la base de datos DB2 de acuerdo con las características de los servicios. En el Servicio uno, se utiliza un bean de entidad:

```
//=================================================
// Este método devuelve la información de un empleado
// mediante la interacción con el bean de entidad localizado
// mediante 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)
{
......
```

La línea uno del código sirve para acceder a la base de datos DB2. En el servicio de visualización de números de empleado, el bean de sesión, AccessEmployee, accede directamente a la base de datos DB2 sample.

```
/==========================================
* Obtener la lista de números de empleado.
* @return Collection
*/
public Collection getEmpNoList()
{
ResultSet rs = null;
PreparedStatement ps = null;
Vector list = new Vector();
DataSource ds = null;
Connection con = null;
try
{
ds = getDataSource();
con = ds.getConnection();
String schema = getEnvProps(DBschema);
String query = "Select EMPNO from " + schema + ".EMPLOYEE";
ps = con.prepareStatement(query);
ps.executeQuery();
rs = ps.getResultSet();
EmployeeKey pk;
while (rs.next())
{
pk = new EmployeeKey();
pk.employeeId = rs.getString(1);
list.addElement(pk.employeeId);
}
rs.close();
return list;
```

El programa de ejemplo `AccessEmployee.ear` utiliza Enterprise Java Beans para implantar una aplicación Java 2 Enterprise Edition para acceder a la base de datos DB2. Encontrará este ejemplo en el directorio `SQLLIB/samples/websphere`.

Consulta relacionada:

- "Programas de ejemplo de Java WebSphere" en el manual *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

WebSphere

Las secciones siguientes describen la colocación en antememoria de sentencias y la agrupación de conexiones WebSphere.

Conexión con datos de la empresa

Con compañías que confían cada vez más en sus datos almacenados, es necesario tenerlos todos en sistemas grandes como servidores zSeries. Con esta nueva necesidad de consolidación, las aplicaciones Web necesitan formas de acceder a los datos de la empresa.

DB2 Connect ofrece a las aplicaciones basadas en Windows o en UNIX la posibilidad de conectar con datos almacenados en estos sistemas grandes y de poderlos utilizar. DB2 también ofrece su propio conjunto de funciones como la agrupación de conexiones y el concentrador de conexiones.

Fuentes de datos y agrupación de conexiones WebSphere

Cada vez que un recurso intenta acceder a una base de datos, debe establecer conexión con dicha base de datos. Una conexión con una base de datos implica actividad general; necesita recursos para crear la conexión, mantenerla y luego liberarla cuando ya no la necesita.

Nota: la información que se proporciona aquí se refiere a la Versión 4 de WebSphere Application Server para UNIX, LINUX y Windows.

La actividad general total de la base de datos correspondiente a una aplicación es especialmente alta para aplicaciones basadas en la Web, porque los usuarios de la Web se conectan y desconectan con más frecuencia. Además, las interacciones de los usuarios suelen ser más cortas, debido a la naturaleza de Internet. Normalmente, se emplea más esfuerzo en conectar y desconectar que durante la propia transacción. Además, puesto que las peticiones de Internet pueden llegar prácticamente de cualquier lugar, los volúmenes de uso pueden ser superiores y más difíciles de predecir.

IBM WebSphere Application Server permite a los administradores establecer una agrupación de conexiones de bases de datos que pueden compartir las aplicaciones de un servidor de aplicaciones para solucionar estos problemas de actividad general.

La agrupación de conexiones extiende la actividad general de conexión entre varias peticiones de usuarios, por lo que conserva los recursos para futuras peticiones.

Puede utilizar la agrupación de conexiones de WebSphere o el soporte de agrupación de conexiones de DB2, que se proporciona mediante la API Paquete opcional de JDBC 2.1, para establecer la agrupación de conexiones.

La agrupación de conexiones de WebSphere es la implantación de la especificación de la API Paquete opcional de JDBC 2.1. Por lo tanto, el modelo de programación de agrupación de conexiones es el que se especifica en las especificaciones JDBC 2.1 Core y API Paquete opcional de JDBC 2.1. Esto significa que las aplicaciones que obtienen sus conexiones a través de una fuente de datos creada en WebSphere Application Server pueden aprovechar las funciones de JDBC 2.1 como la agrupación de conexiones y las conexiones habilitadas para JTA.

Además, la agrupación de conexiones de WebSphere proporciona funciones adicionales que permiten a los administradores ajustar la agrupación para optimizar su rendimiento y proporcionar aplicaciones con excepciones de WebSphere que permiten a los programadores escribir aplicaciones sin conocer las SQLExceptions comunes específicas del proveedor. No todas las SQLExceptions específicas del proveedor se correlacionan con excepciones de WebSphere; se deben codificar las aplicaciones para que puedan manejar las SQLExceptions específicas del proveedor. Sin embargo, las excepciones recuperables más comunes se correlacionan con excepciones de WebSphere.

La fuente de datos obtenida a través de WebSphere es una fuente de datos que implanta la API Paquete opcional de JDBC 2.1. Proporciona agrupación de conexiones y, en función de la fuente de datos seleccionada específica del proveedor, es posible que proporcione conexiones capaces de participar en transacciones de protocolo de confirmación en dos fases (habilitadas para JTA).

El programa `AccessEmployee` del archivo `AccessEmployee.ear` utiliza `DataSource` de WebSphere para acceder a una base de datos DB2.

Conceptos relacionados:

- “Restricciones de la API central de JDBC 2.1 por parte del controlador JDBC de DB2 de tipo 2” en la página 299
- “Soporte de la API de paquete opcional de JDBC 2.1 por parte del controlador de JDBC de DB2 de tipo 2” en la página 300

Consulta relacionada:

- “Programas de ejemplo de Java WebSphere” en el manual *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

Parámetros para ajustar las agrupaciones de conexiones de WebSphere

Se pueden realizar mejoras en el rendimiento ajustando correctamente los parámetros en la agrupación de conexiones. Esta sección detalla cada una de las propiedades que se encuentran en la pestaña Agrupación de conexiones y cómo se pueden ajustar para conseguir un rendimiento óptimo.

Las siguientes propiedades se encuentran en la pestaña Agrupación de conexiones de la ventana de creación de fuente de datos y en la pestaña Agrupación de conexiones de la ventana de propiedades de la fuente de datos:

Tamaño mínimo de la agrupación
El número mínimo de conexiones que puede albergar la agrupación de conexiones. Por omisión, este valor es 1. Cualquier entero no negativo es un valor válido para esta propiedad. El tamaño mínimo de la agrupación puede afectar al rendimiento de una aplicación. Agrupaciones menores requieren menos actividad general cuando la demanda es baja, porque se mantienen abiertas menos conexiones con la base de datos. Por otro lado, cuando la demanda es alta, las primeras aplicaciones obtendrán una respuesta lenta porque se tendrán que crear nuevas conexiones si se están utilizando las de la agrupación.

Tamaño máximo de la agrupación
El número máximo de conexiones que puede albergar la agrupación de conexiones. Por omisión, este valor es 10. Cualquier entero positivo es un valor válido para esta propiedad. El tamaño máximo de la agrupación puede afectar al rendimiento de una aplicación. Agrupaciones mayores requieren más actividad general cuando la demanda es alta, porque hay más conexiones abiertas con la base de datos en momentos de demanda punta. Estas conexiones permanecen hasta que quedan desocupadas fuera de la agrupación. Por otro lado, si el máximo es menor, pueden producirse tiempos de espera más largos o errores de tiempo de espera excedido de conexión durante momentos de demanda punta. La base de datos debe ser capaz de dar soporte al número máximo de conexiones en el servidor de aplicaciones, además de a cualquier carga que pueda tener fuera del servidor de aplicaciones.

Tiempo de espera excedido de conexión
El número máximo de segundos que una aplicación espera una conexión procedente de la aplicación antes de exceder el tiempo de espera enviando una ConnectionWaitTimeoutException a la aplicación. El valor por omisión es 180 segundos (3 minutos). Cualquier entero no negativo es un valor válido para esta propiedad. Si establece para esta propiedad el valor 0, se inhabilita el tiempo de espera excedido de la conexión.

Tiempo de espera excedido de desocupación
EL número de segundos que una conexión puede estar libre en la agrupación antes de que la conexión se elimine de la agrupación. El valor por omisión es 1800 segundos (30 minutos). Cualquier entero no negativo es un valor válido para esta propiedad. Las conexiones tienen que estar desocupadas fuera de la agrupación, porque

mantener conexiones abiertas con la base de datos puede ocasionar problemas de memoria con la base de datos. Sin embargo, no todas las conexiones quedan desocupadas fuera de la agrupación, aunque superen el valor especificado para la propiedad Tiempo de espera excedido de desocupación. Una conexión no queda desocupada si al eliminar la conexión la agrupación quedaría con un tamaño menor al de su tamaño mínimo. Si establece para esta propiedad el valor 0, se inhabilita el tiempo de espera excedido de desocupación.

Tiempo de espera excedido de orfandad

El número de segundos que una aplicación puede mantener una conexión inactiva. El valor por omisión es 1800 segundos (30 minutos). Cualquier entero no negativo es un valor válido para esta propiedad. Si no ha habido actividad en una conexión asignada durante más tiempo del especificado en el valor Tiempo de espera excedido de orfandad, la conexión se marca como huérfana. Transcurridos otra vez el número de segundos especificados en Tiempo de espera excedido de orfandad, si la conexión sigue sin actividad, se devuelve a la agrupación. Si la aplicación vuelve a intentar utilizar la conexión, recibe una StaleConnectionException. Las conexiones que aparecen listadas en una transacción no son huérfanas. Si establece para esta propiedad el valor 0, se inhabilita el tiempo de espera excedido de orfandad.

Tamaño de antememoria de sentencias

El número de sentencias preparadas colocadas en antememoria se mantiene para una agrupación de conexiones entera. El valor por omisión es 100. Cualquier entero no negativo es un valor válido para esta propiedad. Cuando una sentencia se coloca en antememoria, se aumenta el rendimiento, porque una sentencia se recupera de la antememoria si se encuentra una sentencia coincidente, en lugar de tener que crear una nueva sentencia preparada (lo que constituye una operación más cara). El tamaño de antememoria de sentencias no cambia el modelo de programación, únicamente el rendimiento de la aplicación. El tamaño de antememoria de sentencias es el número de sentencias colocadas en antememoria correspondientes a la agrupación entera, no a cada conexión. Poder establecer el tamaño de antememoria de sentencias en la consola administrativa constituye una nueva opción de la Versión 4.0. En versiones anteriores, este valor sólo se podía establecer mediante un archivo datasources.xml. El uso de datasources.xml no se recomienda en la Versión 4.0.

Inhabilitar limpieza automática de conexiones

Indica si la conexión se tiene o no que cerrar al final de una transacción. El valor por omisión es false, lo que indica que, una vez completada una transacción, WebSphere cierra la conexión y la devuelve a la agrupación. Esto significa que, si se intenta utilizar la

conexión una vez finalizada la transacción, se recibe una `StaleConnectionException`, porque la conexión se ha cerrado y ha vuelto a la agrupación. Este mecanismo asegura que la aplicación no mantiene conexiones de forma indefinida. Si el valor es true, la conexión no se devuelve a la agrupación al final de una transacción. En este caso, la aplicación debe devolver la conexión a la agrupación llamando al método `close()`. Si la aplicación no cierra la conexión, la agrupación puede quedarse sin conexiones para que las utilicen otras aplicaciones.

Cada uno de estos parámetros configurables puede determinar el modo en que se utilizan los recursos en cada agrupación. Los dos parámetros más importantes son Tamaño mínimo de la agrupación y Tamaño máximo de la agrupación. A continuación se muestran definiciones más detalladas y posibilidades de uso de estos dos parámetros:

Tamaño mínimo de la agrupación

El número mínimo de conexiones que la agrupación de conexiones puede mantener abiertas con la base de datos. El valor por omisión es 1. En las versiones 3.5 y 4.0, la agrupación no crea el número mínimo de conexiones con la base de datos en un principio. En su lugar, a medida que se necesitan conexiones adicionales, se crean nuevas conexiones con la base de datos, con lo que la agrupación va creciendo. Cuando la agrupación alcanza el número mínimo de conexiones, no se reduce por debajo de este mínimo.

El valor mínimo correcto para la agrupación se puede determinar examinando las aplicaciones que utilizan la agrupación. Si, por ejemplo, se determina que se necesitan al menos cuatro conexiones en cualquier momento, el número mínimo de conexiones se debe establecer en 4 para asegurar que todas las peticiones se pueden responder sin excepciones de tiempo de espera excedido de conexión. En momentos de baja demanda, la agrupación recupera su número mínimo de conexiones. Una buena norma general a seguir consiste en mantener este número tan bajo como sea posible para evitar mantener conexiones abiertas innecesarias.

Tamaño máximo de la agrupación

El número máximo de conexiones que la agrupación de conexiones puede mantener abiertas con la base de datos. La agrupación alberga este número máximo de conexiones abiertas con la base de datos en momentos punta. En momentos de baja demanda, la agrupación vuelve al número mínimo de conexiones.

Lo más recomendable es asegurarse de que sólo se necesite una conexión en una hebra en cualquier momento. Esto evita posibles puntos muertos cuando la agrupación está a su capacidad máxima y no queda ninguna conexión para responder a una petición de

conexión. Por lo tanto, con una conexión por hebra, el tamaño máximo de la agrupación se puede establecer en el número máximo de hebras.

Cuando se utilizan servlets, esto se puede determinar examinando la propiedad `MaxConnections` del Motor de servlets. Si se necesitan varias conexiones en una hebra, el valor del tamaño máximo de la conexión se puede determinar mediante la siguiente fórmula:

```
T *(C -1)+1
```

Donde T es el número de hebras y C es el número de conexiones.

Los otros parámetros de ajuste dependen en gran medida de la aplicación si se utilizan para llevar a cabo ajustes. En cada caso es importante saber aproximadamente durante cuánto tiempo la aplicación promedio utilizará una conexión de la agrupación. Por ejemplo, si se sabe que todas las aplicaciones que utilizan una agrupación de conexiones mantienen una conexión durante un tiempo medio de 5 segundos, con un máximo de 10 segundos, puede resultar útil establecer para Tiempo de espera excedido de orfandad el valor 10 ó 15 segundos, pero estar preparados para recibir alguna `StaleConnectionException` ocasional.

No se recomienda definir deliberadamente las conexiones como huérfanas, aunque puede resultar útil en algunos escenarios de determinación de problemas en los que las conexiones se mantienen durante periodos de tiempo largos y se sabe que la aplicación sólo necesita una conexión durante un periodo corto.

Tiempo de espera excedido de inactividad es un parámetro útil si utiliza una máquina con recursos enlazados. Si ha definido correctamente los tamaños mínimo y máximo de la agrupación de conexiones, es posible que desee reducir el Tiempo de espera excedido de inactividad de modo que, cuando el uso de la agrupación sea bajo, no haya conexiones abiertas sin actividad. Tenga cuidado al reducir el valor de este parámetro porque establecer un valor demasiado bajo genera el coste de crear conexiones con más aplicaciones cuando comienza la transformación de carga ligera a carga pesada.

Finalmente, el Tiempo de espera excedido de conexión se puede utilizar para asegurar que las aplicaciones no esperan indefinidamente una conexión. Si se sabe que todas las aplicaciones que utilizan una agrupación utilizan conexiones durante pocos segundos, puede resultar útil reducir este parámetro; sin embargo, no establezca un valor demasiado bajo para los sistemas con carga pesada. Por ejemplo, si se sabe que la mayoría de las aplicaciones que utilizan una agrupación utilizan una conexión durante al menos 10 segundos (consultas de larga ejecución), cuando este sistema tiene una carga por debajo de la media es posible que las aplicaciones se copien por

detrás de otra que espera conexiones de la agrupación. Cuando más se tarde en volver a colocar una aplicación en línea, más posibilidades hay de que se obtenga un Tiempo de espera excedido de conexión. Tenga en cuenta que esto es un caso raro, puesto que no muchas aplicaciones de la web tienen consultas de larga ejecución a las que intenten accede muchos usuarios simultáneamente. Tenga también en cuenta que, si se modifican los parámetros Tamaño mínimo y máximo de la agrupación de conexiones, puede que no vuelva a tener el problema de copia de seguridad.

Como en el último ejemplo, es importante tener en cuenta todos los parámetros de ajuste cuando se ajuste el sistema. Si sólo se modifican el tamaño mínimo y máximo de la agrupación se puede mejorar el rendimiento, pero puede no resultar de ayuda en la contención de recursos si hay otros parámetros definidos de forma incorrecta para el sistema.

Como sucede en la mayoría de los casos de ajuste, es recomendable probar distintos valores y ver qué combinación funciona mejor para el sistema.

Ventajas de la agrupación de conexiones de WebSphere

La agrupación de conexiones puede mejorar el tiempo de respuesta de cualquier aplicación que necesite conexiones, especialmente aplicaciones basadas en la Web.

Cuando un usuario realiza una petición a un recurso sobre la Web, el recurso accede a una fuente de datos. La mayoría de las peticiones de usuario no implican la actividad general de crear una nueva conexión, porque la fuente de datos puede localizar y utilizar una conexión existente de la agrupación de conexiones. Cuando la petición se responde y la respuesta se devuelve al usuario, el recurso devuelve la conexión a la agrupación de conexiones para que se reutilice. De nuevo, se evita la actividad general de una desconexión.

Cada usuario asimila una facción del coste de conexión o desconexión. Una vez se han utilizado los recursos iniciales para generar las conexiones de la agrupación, la actividad general adicional es insignificante porque se reutilizan las conexiones existentes.

La colocación en antememoria de sentencias preparadas es otro mecanismo por el cual el método de agrupación de conexiones de WebSphere mejora los tiempos de respuesta de aplicaciones basadas en la Web.

Una antememoria de sentencias preparadas previamente está disponible en una conexión. Cuando se solicita una nueva sentencia preparada en una conexión, se devuelve la sentencia preparada colocada en antememoria, si está disponible. Esta colocación en antememoria reduce el número de sentencias preparadas creadas, que resultan caras, lo cual mejora los tiempos de

respuesta. La antememoria resulta útil para aplicaciones que tienden a preparar la misma sentencia varias veces.

Además de mejorar los tiempos de respuesta, el método de agrupación de conexiones de WebSphere proporciona una capa de abstracción de la base de datos, que puede colocar en almacenamiento intermedio la aplicación cliente y hacer que parezca que distintas bases de datos funcionan del mismo modo ante una aplicación.

Esta colocación en almacenamiento intermedio facilita la conmutación de bases de datos de aplicación, puesto que el código de la aplicación no tiene que manejar SQLExceptions comunes específicas del cliente, sino una excepción de agrupación de conexiones de WebSphere.

Colocación de sentencias en antememoria en WebSphere

WebSphere proporciona un mecanismo para colocar en antememoria sentencias preparadas previamente. La colocación en antememoria de sentencias preparadas mejora los tiempos de respuesta, ya que una aplicación puede reutilizar una `PreparedStatement` en una conexión si existe en la antememoria de dicha conexión, sin tener que crear una nueva `PreparedStatement`.

Cuando una aplicación crea una `PreparedStatement` en una conexión, primero se busca en la antememoria de la conexión para determinar si ya existe una `PreparedStatement` con la misma serie SQL. Esta búsqueda se realiza utilizando la serie entera de sentencias de SQL en el método `prepareStatement()`. Si se encuentra una coincidencia, se devuelve la PreparedStatement colocada en antememoria para que se pueda utilizar. Si no se encuentra, se crea una nueva `PreparedStatement` y se devuelve a la aplicación.

A medida que la aplicación cierra sentencias preparadas, estas se devuelven a la antememoria de sentencias de la conexión. Por omisión, sólo se pueden mantener 100 sentencias preparadas en antememoria para la agrupación entera de conexiones. Por ejemplo, si hay diez conexiones en la agrupación, el número de sentencias preparadas colocadas en antememoria correspondientes a estas diez conexiones no puede ser mayor que 100. Esto asegura que hay un número limitado de sentencias preparadas simultáneamente abiertas para la base de datos, lo que ayuda a evitar problemas de recursos con una base de datos.

Los elementos sólo se eliminan de la antememoria de sentencias preparadas de la conexión cuando el número de sentencias preparadas colocadas simultáneamente en antememoria supera el valor de `statementCacheSize` (que por omisión es 100). Si una sentencia preparada se tiene que eliminar de la

antememoria, se elimina y se añade a un vector de sentencias descartadas. En cuanto finaliza el método en el que se ha eliminado la sentencia preparada, las sentencias preparadas del vector de sentencias descartadas se cierran para la base de datos. Por lo tanto, en un determinado momento, puede haber 100 más el número de sentencias recientemente descartadas abiertas para la base de datos. Las sentencias preparadas adicionales se cierran cuando el método finaliza.

El número de sentencias preparadas que se pueden colocar en antememoria se puede configurar en la fuente de datos. Cada antememoria se debe ajustar según los requisitos de la aplicación en cuanto a sentencias preparadas.

Parte 4. Otras interfaces de programación

Capítulo 12. Programación en Perl

Consideraciones sobre la programación en Perl

Perl es un lenguaje de programación popular que está disponible libremente para muchos sistemas operativos. Utilizando el controlador DBD::DB2 disponible en http://www.ibm.com/software/data/db2/perl con el Módulo de Interfaz de base de datos (DBI) de Perl disponible en http://www.perl.com, puede crear aplicaciones DB2 mediante Perl.

Dado que Perl es un lenguaje interpretado y que el Módulo DBI de Perl utiliza SQL dinámico, Perl constituye el lenguaje ideal para crear y revisar con rapidez los prototipos de aplicaciones DB2. El Módulo DBI de Perl utiliza una interfaz que es muy parecida a las interfaces CLI y JDBC, lo cual facilita el transporte de los prototipos de Perl a CLI y JDBC.

La mayoría de proveedores de bases de datos proporcionan un controlador de base de datos para el Módulo DBI de Perl, lo cual significa que también se puede utilizar Perl para crear aplicaciones que accedan a datos de muchos servidores de bases de datos distintos. Por ejemplo, puede escribir en Perl una aplicación DB2 que conecte con una base de datos Oracle utilizando el controlador de base de datos DBD::Oracle, captar datos de la base de datos Oracle e insertar los datos en una base de datos DB2 utilizando el controlador de base de datos DBD::DB2.

Restricciones de Perl

El módulo DBI de Perl sólo soporta SQL dinámico. Cuando sea necesario ejecutar una sentencia varias veces, se puede mejorar el rendimiento de las aplicaciones DB2 en Perl emitiendo una llamada `prepare` para preparar la sentencia.

Para obtener información actual sobre las restricciones de la versión del controlador de DBD::DB2 que instale en la estación de trabajo, consulte el archivo CAVEATS contenido en el paquete del controlador de DBD::DB2.

Acceso a bases de datos de varias hebras en Perl

Perl no da soporte al acceso a bases de datos de varias hebras.

Conexiones de bases de datos en Perl

Para permitir que Perl cargue el módulo DBI, debe incluir la línea siguiente en la aplicación DB2:

```
use DBI;
```

El módulo DBI carga automáticamente el controlador DBD::DB2 cuando se crea un *descriptor de contexto de base de datos* utilizando la sentencia `DBI->connect` con la sintaxis siguiente:

```
my $dbhandle = DBI->connect('dbi:DB2:dbalias', $userID, $password);
```

donde:

$dbhandle
: representa el descriptor de contexto de base de datos devuelto por la sentencia de conexión

dbalias
: representa un alias de DB2 catalogado en el directorio de bases de datos de DB2

$userID
: representa el ID de usuario utilizado para conectar con la base de datos

$password
: representa la contraseña para el ID de usuario utilizado para conectar con la base de datos

Captación de resultados en Perl

Puesto que el Módulo DBI de Perl sólo soporta SQL dinámico, no debe utilizar variables del lenguaje principal en las aplicaciones DB2 en Perl.

Procedimiento:

Para devolver resultados de una consulta de SQL, lleve a cabo los pasos siguientes:

1. Cree un descriptor de contexto de base de datos estableciendo conexión con la base de datos con la sentencia `DBI->connect`.

2. Cree un descriptor de contexto de sentencia a partir del descriptor de contexto de base de datos. Por ejemplo, puede llamar a prepare con una sentencia de SQL como argumento de serie para devolver el descriptor de contexto de sentencia *$sth*, tal como se muestra en la sentencia de Perl siguiente:

```
my $sth = $descripbd->prepare(
   'SELECT firstnme, lastname
      FROM employee '
   );
```

3. Ejecute la sentencia de SQL llamando a execute en el descriptor de contexto de sentencia. Una llamada satisfactoria a execute asocia un conjunto de resultados al descriptor de contexto de sentencia. Por ejemplo, puede ejecutar la sentencia preparada en el ejemplo anterior utilizando la sentencia de Perl siguiente:

```
#Nota: $rc representa el código de retorno de la llamada a execute
my $rc = $sth->execute();
```

4. Capte una fila del conjunto de resultados asociado al descriptor de contexto de sentencia mediante una llamada a fetchrow(). El DBI de Perl devuelve una fila en forma de matriz con un valor por columna. Por ejemplo, puede devolver todas las filas del descriptor de contexto de sentencia del ejemplo anterior utilizando la sentencia de Perl siguiente:

```
while (($firstnme, $lastname) = $sth->fetchrow()) {
   print "$firstnme $lastname\n";
}
```

Conceptos relacionados:

• "Conexiones de bases de datos en Perl" en la página 364

Marcadores de parámetros en Perl

Para permitirle ejecutar una sentencia preparada utilizando distintos valores de entrada para campos específicos, el módulo DBI de Perl le permite preparar y ejecutar una sentencia utilizando marcadores de parámetros. Para incluir un marcador de parámetro en una sentencia de SQL, utilice un signo de interrogación (?).

El código Perl siguiente crea un descriptor de contexto de sentencia que acepta un marcador de parámetro para la cláusula WHERE de una sentencia SELECT. A continuación, el código ejecuta la sentencia dos veces, utilizando los valores de entrada 25000 y 35000 para sustituir el marcador de parámetro.

```
my $sth = $descripbd->prepare(
   'SELECT firstnme, lastname
      FROM employee
      WHERE salary > ?'
   );
```

```
my $rc = $sth->execute(25000);
⋮
my $rc = $sth->execute(35000);
```

Variables SQLSTATE y SQLCODE en Perl

Para devolver el SQLSTATE asociado a un descriptor de contexto de base de datos del DBI de Perl o a un descriptor de contexto de sentencia, llame al método `state`. Por ejemplo, para devolver el SQLSTATE asociado al descriptor de contexto de base de datos $descripbd, incluya la sentencia de Perl siguiente en la aplicación:

```
my $sqlstate = $descripbd->state;
```

Para devolver el SQLCODE asociado a un descriptor de contexto de base de datos del DBI de Perl o a un descriptor de contexto de sentencia, llame al método `err`. Para devolver el mensaje para un SQLCODE asociado a un descriptor de contexto de base de datos del DBI de Perl o a un descriptor de contexto de sentencia, llame al método `errstr`. Por ejemplo, para devolver el SQLCODE asociado al descriptor de contexto de base de datos $descripbd, incluya la sentencia de Perl siguiente en la aplicación:

```
my $sqlcode = $descripbd->err;
```

Ejemplo de programa Perl

A continuación se muestra un ejemplo de una aplicación escrita en Perl:

```
#!/usr/bin/perl
use DBI;

my $database='dbi:DB2:sample';
my $user='';
my $password='';

my $dbh = DBI->connect($database, $user, $password)
   or die "Can't connect to $database: $DBI::errstr";

my $sth = $dbh->prepare(
   q{ SELECT firstnme, lastname
      FROM employee }
   )
   or die "Can't prepare statement: $DBI::errstr";

my $rc = $sth->execute
   or die "Can't execute statement: $DBI::errstr";

print "Query will return $sth->{NUM_OF_FIELDS} fields.\n\n";
print "$sth->{NAME}->[0]: $sth->{NAME}->[1]\n";
```

```
while (($firstnme, $lastname) = $sth->fetchrow()) {
   print "$firstnme: $lastname\n";
}

# comprobar si hay problemas que puedan haber cancelado antes la captación
warn $DBI::errstr if $DBI::err;

$sth->finish;
$dbh->disconnect;
```

Capítulo 13. Programación en REXX

Consideraciones sobre la programación en REXX

En las secciones siguientes se explican las consideraciones especiales sobre la programación en el lenguaje principal. Se incluye información sobre la incorporación de sentencias de SQL, las restricciones del lenguaje y los tipos de datos soportados para las variables del lenguaje principal.

Nota: soporte de REXX estabilizado en DB2 Versión 5, y no hay ninguna mejora de REXX planificada para el futuro. Por ejemplo, REXX no puede manejar identificadores de objetos SQL, como por ejemplo nombres de tabla, que tengan una longitud superior a 18 bytes. Para utilizar las características incorporadas en DB2 después de la Versión 5, como por ejemplo los nombres de tabla de longitud entre 19 y 128 bytes, debe escribir sus aplicaciones en un lenguaje que no sea REXX.

Puesto que REXX es un lenguaje interpretado, no se utiliza ningún precompilador, compilador ni enlazador. En su lugar, se utilizan tres API de DB2 para crear aplicaciones DB2 en REXX. Utilice dichas API para acceder a distintos elementos de DB2.

SQLEXEC
Da soporte al lenguaje SQL.

SQLDBS
Da soporte a las versiones de línea de mandatos de las API de DB2.

SQLDB2
Da soporte a una interfaz específica de REXX con el procesador de línea de mandatos. Consulte la descripción de la sintaxis de la API para REXX para ver detalles y restricciones sobre cómo se puede utilizar esta interfaz.

Conceptos relacionados:

- "Sintaxis de las API para REXX" en la página 385

Restricciones del lenguaje en REXX

Las siguientes secciones describen las restricciones del lenguaje correspondientes a REXX.

Restricciones del lenguaje en REXX

Es posible que los símbolos que se encuentren dentro de sentencias o mandatos que se pasen a las rutinas SQLEXEC, SQLDBS y SQLDB2 puedan corresponder a variables de REXX. En este caso, el intérprete de REXX sustituye el valor de la variable antes de llamar a SQLEXEC, SQLDBS o SQLDB2.

Para evitar esta situación, encierre las series de sentencias entre comillas (' ' o " "). Si no utiliza comillas, el intérprete de REXX resolverá los nombres de variable conflictivos, en lugar de que se pasen a las rutinas SQLEXEC, SQLDBS o SQLDB2.

En REXX/SQL no se soporta el SQL compuesto.

Los procedimientos almacenados de REXX/SQL se soportan en los sistemas operativos Windows, pero no en AIX.

Tareas relacionadas:

- "Registro de SQLEXEC, SQLDBS y SQLDB2 en REXX" en la página 370

Registro de SQLEXEC, SQLDBS y SQLDB2 en REXX

Antes de utilizar cualquiera de las API de DB2 o emitir sentencias de SQL en una aplicación, debe registrar las rutinas SQLDBS, SQLDB2 y SQLEXEC. Así se notifica al intérprete de REXX sobre los puntos de entrada de REXX/SQL.

El método que utilice para realizar el registro variará ligeramente entre las plataformas basadas en Windows y AIX.

Procedimiento:

Utilice los ejemplos siguientes para ver la sintaxis correcta para registrar cada rutina:

Registro de ejemplo en plataformas basadas en Windows

```
/* ------------ Registrar SQLDBS con REXX  -------------------------*/
If Rxfuncquery('SQLDBS')  <> 0 then
    rcy = Rxfuncadd('SQLDBS','DB2AR','SQLDBS')
If rcy \= 0 then
    do
      say 'SQLDBS was not successfully added to the REXX environment'
     signal rxx_exit
   end

/* ------------ Registrar SQLDB2 con REXX  -------------------------*/
If Rxfuncquery('SQLDB2')  <> 0 then
    rcy = Rxfuncadd('SQLDB2','DB2AR','SQLDB2')
If rcy \= 0 then
    do
      say 'SQLDB2 was not successfully added to the REXX environment'
     signal rxx_exit
   end

/* ----------------- Registrar SQLEXEC con REXX  --------------------*/
If Rxfuncquery('SQLEXEC') <> 0 then
    rcy = Rxfuncadd('SQLEXEC','DB2AR','SQLEXEC')
If rcy \= 0 then
    do
      say 'SQLEXEC was not successfully added to the REXX environment'
     signal rxx_exit
   end
```

Ejemplo de registro en AIX

```
   /* ------------ Registrar SQLDBS, SQLDB2 y SQLEXEC con REXX --------*/
 rcy = SysAddFuncPkg("db2rexx")
 If rcy \= 0 then
   do
     say 'db2rexx was not successfully added to the REXX environment'
     signal rxx_exit
   end
```

En plataformas basadas en Windows, sólo es necesario ejecutar una vez el mandato RxFuncAdd para todas las sesiones.

En AIX, se debe ejecutar SysAddFuncPkg en cada aplicación REXX/SQL.

En la documentación de REXX para plataformas basadas en Windows y AIX, respectivamente, encontrará detalles sobre las API RXfuncadd y SysAddFuncPkg.

Acceso a bases de datos de varias hebras en REXX

REXX no da soporte al acceso a bases de datos de varias hebras.

Consideraciones sobre EUC en japonés o chino tradicional para REXX

Las aplicaciones REXX no se soportan en los entornos EUC en japonés o chino tradicional.

SQL incorporado en aplicaciones REXX

Las aplicaciones REXX utilizan API que les permiten utilizar la mayoría de las funciones que suministran las API del gestor de bases de datos y SQL. A diferencia de las aplicaciones escritas en un lenguaje compilado, las aplicaciones REXX no se precompilan. En su lugar, un manejador de SQL dinámico procesa todas las sentencias de SQL. Al combinar REXX con estas API que se pueden llamar, tiene acceso a la mayoría de las funciones del gestor de bases de datos. Aunque REXX no da soporte directamente a algunas API utilizando SQL incorporado, se puede acceder a las mismas utilizando el procesador de línea de mandatos de DB2 desde dentro de la aplicación REXX.

Como REXX es un lenguaje de interpretación, es posible que le resulte más fácil desarrollar y depurar los prototipos de aplicación en REXX que en lenguajes principales compilados. Aunque las aplicaciones DB2 codificadas en REXX no proporcionan el rendimiento de las aplicaciones DB2 que utilizan lenguajes compilados, proporcional la posibilidad de crear aplicaciones DB2 sin tener que precompilar, compilar, enlazar o utilizar software adicional.

Utilice la rutina SQLEXEC para procesar todas las sentencias de SQL. Los argumentos de serie de caracteres para la rutina SQLEXEC están formados por los elementos siguientes:
- Palabras clave de SQL
- Identificadores declarados previamente
- Variables del lenguaje principal de sentencia

Realice cada petición pasando una sentencia de SQL válida a la rutina SQLEXEC. Utilice la sintaxis siguiente:

```
CALL SQLEXEC 'sentencia'
```

Las sentencias de SQL se pueden continuar en más de una línea. Cada parte de la sentencia se debe encerrar entre comillas, y se debe delimitar mediante una coma el texto adicional de la sentencia, del modo siguiente:

```
CALL SQLEXEC 'SQL text',
             'additional text',
                  .
                  .
                  .
             'final text'
```

A continuación se muestra un ejemplo de incorporación de una sentencia de SQL en REXX:

```
statement = "UPDATE STAFF SET JOB = 'Clerk' WHERE JOB = 'Mgr'"
CALL SQLEXEC 'EXECUTE IMMEDIATE :statement'
IF ( SQLCA.SQLCODE < 0) THEN
   SAY 'Update Error:  SQLCODE = ' SQLCA.SQLCODE
```

En este ejemplo, el campo SQLCODE de la estructura SQLCA se comprueba para determinar si la actualización ha resultado satisfactoria.

Se aplican las normas siguientes a las sentencias de SQL incorporado:

- Las sentencias de SQL siguientes se pueden pasar directamente a la rutina SQLEXEC:
  - CALL
  - CLOSE
  - COMMIT
  - CONNECT
  - CONNECT TO
  - CONNECT RESET
  - DECLARE
  - DESCRIBE
  - DISCONNECT
  - EXECUTE
  - EXECUTE IMMEDIATE
  - FETCH
  - FREE LOCATOR
  - OPEN
  - PREPARE
  - RELEASE
  - ROLLBACK
  - SET CONNECTION

  Existen otras sentencias de SQL que se deben procesar dinámicamente utilizando las sentencias EXECUTE IMMEDIATE o PREPARE y EXECUTE junto con la rutina SQLEXEC.
- No se pueden utilizar variables del lenguaje principal en las sentencias CONNECT y SET CONNECTION en REXX.
- Los nombres de cursor y los nombres de sentencia están predefinidos del modo siguiente:

de c1 a c100

Nombres de cursor, que van de *c1* a *c50* para los cursores declarados sin la opción WITH HOLD, y de *c51* a *c100* para los cursores declarados utilizando la opción WITH HOLD.

El identificador de nombre de cursor se utiliza para las sentencias DECLARE, OPEN, FETCH y CLOSE. Identifica el cursor utilizado en la petición de SQL.

de s1 a s100

Nombres de sentencia, que van de *s1* a *s100*.

El identificador de nombre de sentencia se utiliza con las sentencias DECLARE, DESCRIBE, PREPARE y EXECUTE.

Se deben utilizar identificadores declarados previamente para los nombres de cursor y de sentencia. No se admiten otros nombres.

- Cuando se declaren cursores, el nombre de cursor y el nombre de sentencia deben corresponder en la sentencia DECLARE. Por ejemplo, si se utiliza *c1* como nombre de cursor, se debe utilizar *s1* como nombre de sentencia.
- No utilice comentarios dentro de una sentencia de SQL.

Variables del lenguaje principal en REXX

Las secciones siguientes describen cómo declarar y utilizar variables del lenguaje principal en programas REXX.

Variables del lenguaje principal en REXX

Las variables del lenguaje principal son variables del lenguaje REXX a las que se hace referencia en las sentencias de SQL. Permiten que una aplicación pase datos de entrada a DB2 y reciba datos de salida de éste. No es necesario que las aplicaciones REXX declaren las variables del lenguaje principal, a excepción de las variables de localizadores de LOB y de referencia de archivos LOB. Los tamaños y tipos de datos de las variables de sistema principal se determinan en el tiempo de ejecución cuando se hace referencia a las variables. Las siguientes secciones describen las normas a seguir para dar nombre y utilizar variables del lenguaje principal.

Conceptos relacionados:

- “Nombres de variables del lenguaje principal en REXX” en la página 375
- “Referencias a variables del lenguaje principal en REXX” en la página 375
- “Variables de indicador en REXX” en la página 375
- “Variables del lenguaje principal de LOB en REXX” en la página 378
- “Borrado de variables del lenguaje principal de LOB en REXX” en la página 380

Consulta relacionada:

- “Variables de REXX predefinidas” en la página 376
- “Sintaxis de las declaraciones de localizador de LOB en REXX” en la página 378
- “Sintaxis de las declaraciones de referencia de archivos LOB en REXX” en la página 379
- “Tipos de datos SQL soportados en REXX” en la página 381

Nombres de variables del lenguaje principal en REXX

Cualquier variable de REXX correctamente denominada se puede utilizar como variable del lenguaje principal. Un nombre de variable puede tener una longitud máxima de 64 caracteres. No termine el nombre con un punto. Un nombre de variable del lenguaje principal puede constar de caracteres alfabéticos, numéricos y los caracteres @, _, !, ., ? y $.

Referencias a variables del lenguaje principal en REXX

El intérprete de REXX examina cada serie que no tiene comillas de un procedimiento. Si la serie representa a una variable en la agrupación actual de variables de REXX, éste sustituye la serie por el valor actual. A continuación se muestra un ejemplo de cómo se puede hacer referencia a una variable del lenguaje principal en REXX:

```
CALL SQLEXEC 'FETCH C1 INTO :cm'
SAY 'Commission = ' cm
```

Para asegurarse de que una serie de caracteres no se convierta a un tipo de datos numérico, encierre la serie entre comillas tal como en el ejemplo siguiente:

```
VAR = '100'
```

REXX establece la variable *VAR* en la serie de caracteres de 3 bytes 100. Si se tienen que incluir comillas como parte de la serie, siga este ejemplo:

```
VAR = "'100'"
```

Cuando se insertan datos numéricos en un campo de tipo CHARACTER, el intérprete de REXX trata los datos numéricos como datos enteros, por lo que deberá concatenar explícitamente las series numéricas y encerrarlas entre comillas.

Variables de indicador en REXX

El tipo de datos de una variable de indicador en REXX es un número sin coma decimal. A continuación se muestra un ejemplo de variable de indicador en REXX que utiliza la palabra clave INDICATOR.

```
CALL SQLEXEC 'FETCH C1 INTO :cm INDICATOR :cmind'
IF ( cmind < 0 )
   SAY 'Commission is NULL'
```

En el ejemplo anterior, se examina que cmind tenga un valor negativo. Si no es negativo, la aplicación puede utilizar el valor devuelto de cm. Si es negativo, el valor captado es NULL y no se debe utilizar cm. En este caso, el gestor de bases de datos no cambia el valor de la variable del lenguaje principal.

Variables de REXX predefinidas

SQLEXEC, SQLDBS y SQLDB2 establecen variables de REXX predefinidas como consecuencia de determinadas operaciones. Estas variables son:

RESULT

Cada operación establece este código de retorno. Los valores posibles son:

n Donde *n* es un valor positivo que indica el número de bytes de un mensaje formateado. La API GET ERROR MESSAGE sola devuelve este valor.

0 Se ha ejecutado la API. La SQLCA de la variable de REXX contiene el estado de terminación de la API. Si SQLCA.SQLCODE es distinto de cero, SQLMSG contiene el mensaje de texto asociado a este valor.

–1 No se dispone de suficiente memoria para completar la API. No se ha devuelto el mensaje solicitado.

–2 SQLCA.SQLCODE se establece en 0. No se ha devuelto ningún mensaje.

–3 SQLCA.SQLCODE contenía un SQLCODE que no era válido. No se ha devuelto ningún mensaje.

–6 No se ha podido crear la variable SQLCA de REXX. Esto indica que no había suficiente memoria disponible o que la agrupación de variables de REXX no estaba disponible por algún motivo.

–7 No se ha podido crear la variable SQLMSG de REXX. Esto indica que no había suficiente memoria disponible o que la agrupación de variables de REXX no estaba disponible por algún motivo.

–8 No se ha podido captar la variable SQLCA.SQLCODE de REXX de la agrupación de variables de REXX.

–9 La variable SQLCA.SQLCODE de REXX se ha truncado durante la captación. La longitud máxima para esta variable es de 5 bytes.

–10 La variable SQLCA.SQLCODE de REXX no se ha podido convertir de ASCII a un entero largo válido.

–11 No se ha podido captar la variable SQLCA.SQLERRML de REXX de la agrupación de variables de REXX.

–12 La variable SQLCA.SQLERRML de REXX se ha truncado durante la captación. La longitud máxima para esta variable es de 2 bytes.

–13 La variable SQLCA.SQLERRML de REXX no se ha podido convertir de ASCII a un entero corto válido.

–14 No se ha podido captar la variable SQLCA.SQLERRMC de REXX de la agrupación de variables de REXX.

–15 La variable SQLCA.SQLERRMC de REXX se ha truncado durante la captación. La longitud máxima para esta variable es de 70 bytes.

–16 No se ha podido establecer la variable de REXX especificada para el texto del error.

–17 No se ha podido captar la variable SQLCA.SQLSTATE de REXX de la agrupación de variables de REXX.

–18 La variable SQLCA.SQLSTATE de REXX se ha truncado durante la captación. La longitud máxima para esta variable es de 2 bytes.

Nota: los valores –8 a –18 sólo los devuelve la API GET ERROR MESSAGE.

SQLMSG

Si SQLCA.SQLCODE es distinto de cero, esta variable contiene el mensaje de texto asociado al código de error.

SQLISL

Nivel de aislamiento. Los valores posibles son:

RR Lectura repetible.
RS Estabilidad de lectura.
CS Estabilidad del cursor. Éste es el valor por omisión.
UR Lectura no confirmada.
NC Sin confirmación. (NC sólo recibe soporte de algunos servidores de sistema principal, AS/400 o iSeries.)

SQLCA

La estructura SQLCA actualizada después de que se procesen las sentencias de SQL y se llame a las API de DB2.

SQLRODA

La estructura SQLDA de entrada/salida para procedimientos

almacenados invocados mediante la sentencia CALL. También es la estructura SQLDA de salida para procedimientos almacenados invocados mediante la API Database Application Remote Interface (DARI).

SQLRIDA
La estructura SQLDA de entrada para procedimientos almacenados invocados mediante la API Database Application Remote Interface (DARI).

SQLRDAT
Una estructura SQLCHAR para procedimientos de servidor invocados mediante la API Database Application Remote Interface (DARI).

Consulta relacionada:

- "SQLCA" en el manual *Administrative API Reference*
- "SQLCHAR" en el manual *Administrative API Reference*
- "SQLDA" en el manual *Administrative API Reference*

Variables del lenguaje principal de LOB en REXX

Cuando se capte una columna LOB en una variable del lenguaje principal REXX, esta se almacenará como una serie simple (es decir, descontada). Esto se maneja del mismo modo que todos los tipos de SQL basados en caracteres (tales como CHAR, VARCHAR, GRAPHIC, LONG, etc.). En la entrada, si el tamaño del contenido de la variable del lenguaje principal es superior a 32 K, o si cumple con otros criterios establecidos más adelante, se le asignará el tipo de LOB apropiado.

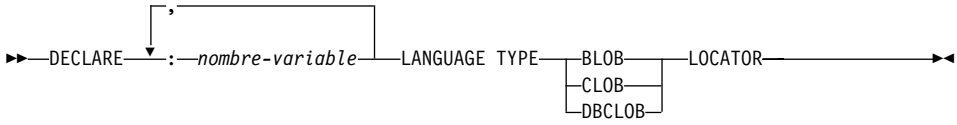
En SQL de REXX, los tipos de SQL se determinan a partir del contenido de la serie de la variable del lenguaje principal, del modo siguiente:

Contenido de la serie de variable del lenguaje principal	Tipo de LOB resultante
:hv1='serie normal entre comillas de más de 32 K ...'	CLOB
:hv2=″'serie con comillas de delimitación incorporadas ″, ″de más de 32 K...'″	CLOB
:hv3=″G'serie DBCS con comillas de delimitación incorporadas, ″, ″que empieza por G, de más de 32 K...'″	DBCLOB
:hv4=″BIN'serie con comillas de delimitación incorporadas, ″, ″que empieza por BIN, de cualquier longitud...'″	BLOB

Sintaxis de las declaraciones de localizador de LOB en REXX

A continuación se muestra la sintaxis para declarar variables del lenguaje principal de localizador de LOB en REXX:

Sintaxis de las variables del lenguaje principal de localizador de LOB en REXX



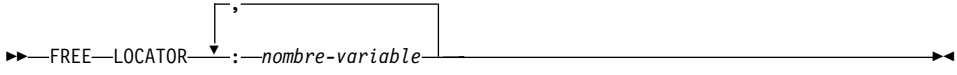
Debe declarar las variables del lenguaje principal de localizador de LOB en la aplicación. Cuando REXX/SQL encuentra estas declaraciones, trata las variables del lenguaje principal declaradas como localizadores durante el resto del programa. Los valores de localizadores se almacenan en variables de REXX, en un formato interno.

Ejemplo:

```
CALL SQLEXEC 'DECLARE :hv1, :hv2 LANGUAGE TYPE CLOB LOCATOR'
```

Los datos representados por localizadores de LOB devueltos por el mecanismo se pueden liberar en REXX/SQL mediante la sentencia FREE LOCATOR, que tiene el formato siguiente:

Sintaxis de la sentencia FREE LOCATOR



Ejemplo:

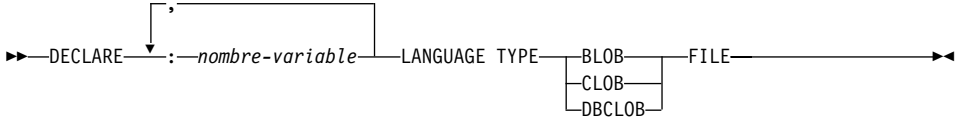
```
CALL SQLEXEC 'FREE LOCATOR :hv1, :hv2'
```

Sintaxis de las declaraciones de referencia de archivos LOB en REXX

Debe declarar las variables del lenguaje principal de referencia de archivos LOB en la aplicación. Cuando REXX/SQL encuentra estas declaraciones, trata las variables del lenguaje principal declaradas como referencias de archivos LOB durante el resto del programa.

A continuación se muestra la sintaxis para declarar variables del lenguaje principal de referencia de archivos LOB en REXX:

Declaraciones de referencia de archivos en REXX



Ejemplo:

```
CALL SQLEXEC 'DECLARE :hv3, :hv4 LANGUAGE TYPE CLOB FILE'
```

Las variables de referencia de archivos en REXX contienen tres campos. Para el ejemplo anterior, éstos son:

hv3.FILE_OPTIONS.
Establecido por la aplicación para indicar cómo se utilizará el archivo.

hv3.DATA_LENGTH.
Establecido por DB2 para indicar el tamaño del archivo.

hv3.NAME.
Establecido por la aplicación con el nombre del archivo LOB.

Para FILE_OPTIONS, la aplicación establece las palabras clave siguientes:

Palabra clave (valor entero)	Significado
READ (2)	El archivo se va a utilizar como entrada. Se trata de un archivo normal que se puede abrir, leer y cerrar. La longitud de los datos del archivo (en bytes) se calcula (lo hace el código solicitante de la aplicación) al abrir el archivo.
CREATE (8)	En la salida, crear un nuevo archivo. Si el archivo ya existe, es un error. La longitud (en bytes) del archivo se devuelve en el campo DATA_LENGTH de la estructura de la variable de referencia de archivos.
OVERWRITE (16)	En la salida, se sobregraba el archivo, si ya existe; de lo contrario, se crea un nuevo archivo. La longitud (en bytes) del archivo se devuelve en el campo DATA_LENGTH de la estructura de la variable de referencia de archivos.
APPEND (32)	La salida se añade al archivo, si existe; de lo contrario, se crea un nuevo archivo. La longitud (en bytes) de los datos que se han añadido al archivo (y no la longitud total del archivo) se devuelve en el campo DATA_LENGTH de la estructura de la variable de referencia de archivos.

Nota: una variable del lenguaje principal de referencia de archivos es una variable compuesta en REXX, por lo que debe establecer valores para los campos NAME, NAME_LENGTH y FILE_OPTIONS además de declararlas.

Borrado de variables del lenguaje principal de LOB en REXX

En plataformas basadas en Windows, puede ser necesario borrar explícitamente las declaraciones de variables del lenguaje principal de referencia de archivos y de localizador de LOB de SQL de REXX, puesto que permanecen en vigor una vez que finaliza el programa de aplicación. Esto es

debido a que el proceso de la aplicación no sale hasta que se cierra la sesión en la que se ejecuta. Si no se borran las declaraciones de LOB del SQL para REXX, pueden interferir con otras aplicaciones que se ejecuten en la misma sesión después de que se haya ejecutado una aplicación de LOB.

La sintaxis para borrar la declaración es:

```
CALL SQLEXEC "CLEAR SQL VARIABLE DECLARATIONS"
```

Debe codificar esta sentencia al final de las aplicaciones de LOB. Observe que la puede codificar en cualquier lugar, como medida de precaución, para borrar las declaraciones que puedan haber quedado de aplicaciones anteriores (por ejemplo, al principio de una aplicación SQL en REXX).

Cursores en REXX

Cuando se declara un cursor en REXX, el cursor se asocia a una consulta. La consulta se asocia a un nombre de sentencia asignado en la sentencia PREPARE. Cualquier variable del lenguaje principal a la que se haga referencia se representa mediante marcadores de parámetros. El siguiente ejemplo muestra una sentencia DECLARE asociada a una sentencia SELECT dinámica:

```
prep_string = "SELECT TABNAME FROM SYSCAT.TABLES WHERE TABSCHEMA = ?"
CALL SQLEXEC 'PREPARE S1 FROM :prep_string';
CALL SQLEXEC 'DECLARE C1 CURSOR FOR S1';
CALL SQLEXEC 'OPEN C1 USING :schema_name';
```

Consulta relacionada:

- “Tipos de datos SQL soportados en REXX” en la página 381

Tipos de datos SQL soportados en REXX

Determinados tipos de datos de REXX predefinidos corresponden a tipos de columna de DB2. La tabla siguiente muestra cómo SQLEXEC y SQLDBS interpretan variables de REXX para convertir su contenido en tipos de datos de DB2.

Nota: no existe soporte de variables del lenguaje principal para el tipo de datos DATALINK en ninguno de los lenguajes principales de DB2.

Tabla 18. Tipos de columna de SQL correlacionados con declaraciones de REXX

Tipo de columna SQL[1]		Tipo de datos de REXX	Descripción del tipo de columna SQL
SMALLINT	(500 ó 501)	Un número sin coma decimal que va de -32 768 a 32 767	Entero con signo de 16 bits
INTEGER (496 ó 497)		Un número sin coma decimal que va de -2 147 483 648 a 2 147 483 647	Entero con signo de 32 bits

Tabla 18. Tipos de columna de SQL correlacionados con declaraciones de REXX (continuación)

Tipo de columna SQL[1]	**Tipo de datos de REXX**	**Descripción del tipo de columna SQL**
REAL[2] (480 ó 481)	Un número en notación científica que va de -3.40282346 x 10^{38} a 3.40282346 x 10^{38}	Coma flotante de precisión única
DOUBLE[3] (480 ó 481)	Un número en notación científica que va de -1.79769313 x 10^{308} a 1.79769313 x 10^{308}	Coma flotante de precisión doble
DECIMAL(*p*,*s*)(484 ó 485)	Un número con una coma decimal	Decimal empaquetado
CHAR(*n*) (452 ó 453)	Una serie con una comilla inicial y otra final (’), cuya longitud es *n* después de eliminar las dos comillas Una serie de longitud *n* sin ningún carácter no numérico, aparte de los blancos inicial y final o la E en notación científica	Serie de caracteres de longitud fija con longitud *n*, donde *n* va de 1 a 254
VARCHAR(*n*) (448 ó 449)	Equivalente a CHAR(*n*)	Serie de caracteres de longitud variable con longitud *n*, donde *n* va de 1 a 4000
LONG VARCHAR (456 ó 457)	Equivalente a CHAR(*n*)	Serie de caracteres de longitud variable con longitud *n*, donde *n* va de 1 a 32 700
CLOB(*n*) (408 ó 409)	Equivalente a CHAR(*n*)	Serie de caracteres de longitud variable y de objeto grande con longitud *n*, donde *n* va de 1 a 2 147 483 647
Variable de localizador CLOB[4] (964 ó 965)	DECLARE :*nombre_var* LANGUAGE TYPE CLOB LOCATOR	Identifica las entidades CLOB que residen en el servidor
Variable de referencia de archivo CLOB[4] (920 ó 921)	DECLARE :*nombre_var* LANGUAGE TYPE CLOB FILE	Descriptor de archivo que contiene datos CLOB
BLOB(*n*) (404 ó 405)	Una serie con un apóstrofo inicial y uno final, precedida de BIN, que contiene *n* caracteres después de eliminar el BIN precedente y los dos apóstrofos.	Serie binaria de longitud variable y de objeto grande con longitud *n*, donde *n* va de 1 a 2 147 483 647
Variable de localizador BLOB[4] (960 ó 961)	DECLARE :*nombre_var* LANGUAGE TYPE BLOB LOCATOR	Identifica las entidades BLOB del servidor
Variable de referencia de archivo BLOB[4] (916 ó 917)	DECLARE :*nombre_var* LANGUAGE TYPE BLOB FILE	Descriptor del archivo que contiene datos BLOB
DATE (384 ó 385)	Equivalente a CHAR(*10*)	Serie de caracteres de 10 bytes
TIME (388 ó 389)	Equivalente a CHAR(*8*)	Serie de caracteres de 8 bytes
TIMESTAMP (392 ó 393)	Equivalente a CHAR(*26*)	Serie de caracteres de 26 bytes

Nota: los tipos de datos siguientes sólo están disponibles en el entorno DBCS.

Tabla 18. Tipos de columna de SQL correlacionados con declaraciones de REXX (continuación)

Tipo de columna SQL[1]	Tipo de datos de REXX	Descripción del tipo de columna SQL
GRAPHIC(*n*) (468 ó 469)	Una serie con un apóstrofo inicial y uno final, precedida de una G o una N, que contiene *n* caracteres DBCS después de eliminar el carácter precedente y los dos apóstrofos	Serie gráfica de longitud fija con longitud *n*, donde *n* va de 1 a 127
VARGRAPHIC(*n*) (464 ó 465)	Equivalente a GRAPHIC(*n*)	Serie gráfica de longitud variable con longitud *n*, donde *n* va de 1 a 2000
LONG VARGRAPHIC (472 ó 473)	Equivalente a GRAPHIC(*n*)	Serie gráfica de longitud variable larga con longitud *n*, donde *n* va de 1 a 16 350
DBCLOB(*n*) (412 ó 413)	Equivalente a GRAPHIC(*n*)	Serie gráfica de longitud variable y de objeto grande con longitud *n*, donde *n* va de 1 a 1 073 741 823
Variable de localizador DBCLOB[4] (968 ó 969)	DECLARE :*nombre_var* LANGUAGE TYPE DBCLOB LOCATOR	Identifica las entidades DBCLOB que residen en el servidor
Variable de referencia de archivos DBCLOB[4] (924 ó 925)	DECLARE :*nombre_var* LANGUAGE TYPE DBCLOB FILE	Descriptor de archivo que contiene datos DBCLOB

Notas:

1. El primer número que se encuentra bajo **Tipo de columna SQL** indica que no se proporciona una variable de indicador, y el segundo número indica que se proporciona una variable de indicador. Se necesita una variable de indicador para indicar los valores nulos (NULL) o para contener la longitud de una serie truncada.
2. FLOAT(*n*) donde $0 < n < 25$ es un sinónimo de REAL. La diferencia entre REAL y DOUBLE en el SQLDA es el valor de la longitud (4 ó 8).
3. Los tipos SQL siguientes son sinónimos de DOUBLE:
   - FLOAT
   - FLOAT(*n*) donde $24 < n < 54$ es
   - DOUBLE PRECISION
4. Éste no es un tipo de columna, sino un tipo de variable del lenguaje principal.

Conceptos relacionados:

- "Cursores en REXX" en la página 381

Requisitos de ejecución para REXX

Las siguientes secciones describen los requisitos de ejecución para aplicaciones REXX.

Creación y ejecución de aplicaciones REXX

Las aplicaciones REXX no se precompilan, compilan ni enlazan. Las instrucciones siguientes describen cómo crear y ejecutar aplicaciones REXX en sistemas operativos Windows y en el sistema operativo AIX.

Restricciones:

En plataformas basadas en Windows, el archivo de aplicación debe tener una extensión .CMD. Después de su creación, puede ejecutar la aplicación directamente desde el indicador de mandatos del sistema operativo. En AIX, el archivo de la aplicación puede tener cualquier extensión.

Procedimiento:

Cree y ejecute las aplicaciones REXX del siguiente modo:

- En sistemas operativos Windows, el archivo de aplicación puede tener cualquier nombre. Después de su creación, puede ejecutar la aplicación desde el indicador de mandatos del sistema operativo invocando al intérprete de REXX del modo siguiente:

  ```
  REXX file_name
  ```

- En AIX, puede ejecutar la aplicación mediante cualquiera de los dos métodos siguientes:
  - En el indicador de mandatos de shell, escriba `rexx name` donde `name` es el nombre del programa REXX.
  - Si la primera línea del programa REXX contiene un "número mágico" (#!) e identifica el directorio en que reside el intérprete de REXX/6000, puede ejecutar el programa REXX escribiendo su nombre en el indicador de mandatos del shell. Por ejemplo, si el archivo del intérprete de REXX/6000 está en el directorio /usr/bin, incluya la información siguiente como primera línea del programa REXX:

    ```
    #! /usr/bin/rexx
    ```

    A continuación, haga que el programa sea ejecutable escribiendo el mandato siguiente en el indicador de mandatos del shell:

    ```
    chmod +x name
    ```

    Ejecute el programa REXX escribiendo su nombre de archivo en el indicador de mandatos del shell.

Nota: en AIX, debe establecer la variable de entorno LIBPATH para incluir el directorio en que está ubicada la biblioteca de SQL para REXX, `db2rexx`. Por ejemplo:

```
export LIBPATH=/lib:/usr/lib:/usr/lpp/db2_08_01/lib
```

Archivos de vinculación de REXX

Se proporcionan cinco archivos de vinculación para el soporte de aplicaciones REXX. Los nombres de estos archivos están incluidos en el archivo

DB2UBIND.LST. Cada archivo de vinculación se ha precompilado utilizando un nivel de aislamiento distinto; por consiguiente, en la base de datos hay cinco paquetes diferentes almacenados.

Los cinco archivos de vinculación son:

DB2ARXCS.BND
Soporta el nivel de aislamiento de estabilidad del cursor.

DB2ARXRR.BND
Soporta el nivel de aislamiento de estabilidad de lectura repetible.

DB2ARXUR.BND
Soporta el nivel de aislamiento de estabilidad de lectura no confirmada.

DB2ARXRS.BND
Soporta el nivel de aislamiento de estabilidad de lectura.

DB2ARXNC.BND
Soporta el nivel de aislamiento de sin confirmación. Se utiliza este nivel de aislamiento cuando se trabaja con algunos servidores de bases de datos de sistema principal, AS/400 o iSeries. En otras bases de datos, se comporta como el nivel de aislamiento de lectura no confirmada.

Nota: en algunos casos, puede que sea necesario vincular de forma explícita estos archivos a la base de datos.

Cuando se utiliza la rutina SQLEXEC se usa, por omisión, el paquete creado con estabilidad del cursor. Si necesita uno de los otros niveles de aislamiento, lo puede cambiar mediante la API SQLDBS CHANGE SQL ISOLATION LEVEL antes de conectar con la base de datos. Esto ocasionará que las posteriores llamadas a la rutina SQLEXEC se asocien al nivel de aislamiento especificado.

Las aplicaciones REXX basadas en Windows no pueden asumir que esté en vigor el nivel de aislamiento por omisión a menos que sepan que ningún otro programa REXX de la sesión ha cambiado el valor. Antes de conectar con una base de datos, una aplicación REXX debe establecer explícitamente el nivel de aislamiento.

Sintaxis de las API para REXX

Utilice la rutina SQLDBS para llamar a las API de DB2 con la sintaxis siguiente:

```
CALL SQLDBS 'command string'
```

Si no puede llamar a una API de DB2 que desea utilizar mediante la rutina SQLDBS, puede llamar a la API efectuando una llamada al procesador de

línea de mandatos (CLP) de DB2 desde dentro de la aplicación REXX. Sin embargo, puesto que el CLP de DB2 dirige la salida al dispositivo de salida estándar o a un archivo especificado, la aplicación REXX no puede acceder directamente a la salida de la API de DB2 a la que se ha llamado, ni puede tomar fácilmente la determinación de si la API llamada ha resultado satisfactoria o no. La API SQLDB2 proporciona una interfaz con el CLP de DB2 que proporciona información directa a la aplicación REXX sobre el éxito o fracaso de cada API llamada, estableciendo la variable compuesta SQLCA de REXX después de cada llamada.

Puede utilizar la rutina SQLDB2 para llamar a las API de DB2 utilizando la sintaxis siguiente:

```
CALL SQLDB2 'command string'
```

donde `'command string'` es una serie que puede procesar el procesador de línea de mandatos (CLP).

El hecho de llamar a una API de DB2 mediante SQLDB2 es equivalente a llamar directamente al CLP, excepto en los aspectos siguientes:

- La llamada al ejecutable del CLP se sustituye por la llamada a SQLDB2 (el resto de opciones y parámetros del CLP se especifican de la misma manera).
- La variable compuesta SQLCA de REXX se establece después de llamar a SQLDB2, pero no se establece después de llamar al ejecutable del CLP.
- La salida de visualización por omisión del CLP se establece como desactivada cuando se llama a SQLDB2, mientras que se establece como salida activada cuando se llama al ejecutable del CLP. Observe que puede activar la salida de visualización del CLP pasando las opciones `+o` o `-o-` a SQLDB2.

Puesto que la única variable de REXX que se establece después de llamar a SQLDB2 es la SQLCA, sólo debe utilizar esta rutina para llamar a las API de DB2 que no devuelvan datos distintos de la SQLCA y que no estén implementadas actualmente mediante la interfaz SQLDBS. Así pues, SQLDB2 únicamente soporta las API de DB2 siguientes:

- Activar base de datos
- Añadir nodo
- Vincular para DB2 Versión 1[(1) (2)]
- Vincular para DB2 Versión 2 ó 5[(1)]
- Crear base de datos en nodo
- Eliminar base de datos en nodo
- Verificar eliminación de nodo
- Desactivar base de datos
- Desregistrar
- Cargar[(3)]

- Cargar consulta
- Precompilar programa(1)
- Revincular paquete(1)
- Redistribuir grupo de particiones de base de datos
- Registrar
- Iniciar gestor de la base de datos
- Detener gestor de la base de datos

Notas sobre las API de DB2 soportadas por SQLDB2:

1. Estos mandatos requieren una sentencia CONNECT a través de la interfaz SQLDB2. Las conexiones que utilizan la interfaz SQLDB2 no están accesibles para la interfaz SQLEXEC, y las conexiones que utilizan la interfaz SQLEXEC no están accesible para la interfaz SQLDB2.
2. Se soporta en plataformas basadas en Windows mediante la interfaz SQLDB2.
3. El parámetro de salida opcional, `pLoadInfoOut` para la API Cargar no se devuelve a la aplicación en REXX.

Nota: aunque la rutina SQLDB2 ha sido pensada para uso únicamente de las API de DB2 relacionadas anteriormente, también se puede utilizar para otras API de DB2 que no se soportan a través de la rutina SQLDBS. Alternativamente, se puede acceder a las API de DB2 a través del CLP desde la aplicación REXX.

Llamada a procedimientos almacenados desde REXX

Las secciones siguientes describen cómo llamar a procedimientos almacenados desde aplicaciones REXX.

Procedimientos almacenados en REXX

Las aplicaciones SQL de REXX pueden llamar a procedimientos almacenados que se encuentren en el servidor de base de datos utilizando la sentencia CALL de SQL. El procedimiento almacenado puede estar escrito en cualquier lenguaje soportado en dicho servidor, a excepción de REXX en los sistemas AIX. (Las aplicaciones cliente pueden estar escritas en REXX en los sistemas AIX, pero, al igual que en otros lenguajes, no pueden llamar a un procedimiento almacenado escrito en REXX en AIX.)

Conceptos relacionados:

- "Llamadas a procedimientos almacenados en REXX" en la página 387

Llamadas a procedimientos almacenados en REXX

La sentencia CALL permite que una aplicación cliente pase datos a un procedimiento almacenado en un servidor, y reciba datos del mismo. La

interfaz para los datos tanto de entrada como de salida es una lista de variables del lenguaje principal. Puesto que REXX suele determinar el tipo y el tamaño de las variables del lenguaje principal en base a su contenido, las variables que sean sólo de salida y se pasen a CALL se deben inicializar con datos *ficticios*, parecidos en tipo y tamaño a la salida esperada.

También se pueden pasar datos a los procedimientos almacenados a través de las variables SQLDA de REXX, mediante la sintaxis USING DESCRIPTOR de la sentencia CALL. La tabla siguiente muestra cómo se establece el SQLDA. En la tabla, ':value' es la raíz de una variable del lenguaje principal REXX que contiene los valores necesarios para la aplicación. Para DESCRIPTOR, 'n' es un valor numérico que indica un elemento *sqlvar* específico del SQLDA. Los números que aparecen a la derecha hacen referencia a las notas que siguen a la tabla.

Tabla 19. SLQDA de REXX de la parte cliente para procedimientos almacenados que utilizan la sentencia CALL

USING DESCRIPTOR	:value.SQLD	1	
	:value.n.SQLTYPE	1	
	:value.n.SQLLEN	1	
	:value.n.SQLDATA	1	2
	:value.n.SQLDIND	1	2

Notas:

1. Antes de invocar al procedimiento almacenado, la aplicación cliente tiene que inicializar la variable de REXX con los datos apropiados.

   Cuando se ejecuta la sentencia CALL de SQL, el gestor de bases de datos asigna almacenamiento y recupera el valor de la variable de REXX de la agrupación de variables de REXX. Para una SQLDA utilizada en una sentencia CALL, el gestor de bases de datos asigna almacenamiento para los campos SQLDATA y SQLIND en base a los valores de SQLTYPE y SQLLEN.

   En el caso de un procedimiento almacenado REXX (es decir, en que el procedimiento al que se llama está escrito en REXX basado en Windows), los datos pasados por el cliente del tipo de sentencia CALL o de la API DARI se colocan en la agrupación de variables de REXX en el servidor de base de datos, utilizando los nombres siguientes predefinidos:

   SQLRIDA
   Nombre predefinido para la variable SQLDA de entrada de REXX

   SQLRODA
   Nombre predefinido para la variable SQLDA de salida de REXX

2. Cuando termina el procedimiento almacenado, el gestor de bases de datos recupera también el valor de las variables del procedimiento almacenado. Los valores se devuelven a la aplicación cliente y se colocan en la agrupación de variables de REXX del cliente.

Conceptos relacionados:

- "Consideraciones del cliente para llamar a procedimientos almacenados en REXX" en la página 389
- "Consideraciones del servidor para llamar a procedimientos almacenados en REXX" en la página 389
- "Recuperación de valores de precisión y escala (SCALE) de campos decimales del SQLDA" en la página 390

Consulta relacionada:

- "CALL sentencia" en el manual *Consulta de SQL, Volumen 2*

Consideraciones del cliente para llamar a procedimientos almacenados en REXX

Cuando utilice variables del lenguaje principal en la sentencia CALL, inicialice cada una de las variables del lenguaje principal con un valor que sea de un tipo compatible con los datos que se devuelvan a la variable del lenguaje principal desde el procedimiento del servidor. Debe realizar esta inicialización aunque el indicador correspondiente sea negativo.

Cuando utilice descriptores, SQLDATA debe estar inicializada y contener datos que sean de un tipo compatible con los datos que se devuelvan desde el procedimiento del servidor. Debe realizar esta inicialización aunque el campo SQLIND contenga un valor negativo.

Consulta relacionada:

- "Tipos de datos SQL soportados en REXX" en la página 381

Consideraciones del servidor para llamar a procedimientos almacenados en REXX

Asegúrese de que todos los campos SQLDATA y SQLIND (si es de un tipo anulable) de la SQLRODA de SQLDA de salida predefinida estén inicializados. Por ejemplo, si SQLRODA.SQLD es 2, los campos siguientes deben contener datos (aunque los indicadores correspondientes sean negativos y no se pasen datos al cliente):

- SQLRODA.1.SQLDATA
- SQLRODA.2.SQLDATA

Recuperación de valores de precisión y escala (SCALE) de campos decimales del SQLDA

Para recuperar los valores de precisión y escala para campos decimales de la estructura SQLDA devuelta por el gestor de bases de datos, utilice los valores `sqllen.scale` y `sqllen.precision` cuando inicialice la salida del SQLDA en el programa REXX. Por ejemplo:

```
 .
 .
 .
/* INICIALIZAR UN ELEMENTO DE SQLDA DE SALIDA */
io_sqlda.sqld = 1
io_sqlda.1.sqltype = 485           /* TIPO DE DATOS DECIMAL */
io_sqlda.1.sqllen.scale  = 2       /* DÍGITOS A LA DERECHA DE LA COMA DECIMAL */
io_sqlda.1.sqllen.precision  = 7   /* ANCHURA DEL DECIMAL  */
io_sqlda.1.sqldata = 00000.00      /* FORMATO DE DATOS DE DEFINICIÓN DE AYUDAS */
io_sqlda.1.sqlind = -1             /* SIN DATOS DE ENTRADA */
 .
 .
 .
```

Capítulo 14. Cómo escribir aplicaciones mediante IBM OLE DB Provider para servidores DB2

Objetivo de IBM OLE DB Provider para DB2

Microsoft OLE DB es un conjunto de interfaces OLE/COM que proporciona a las aplicaciones acceso uniforme a datos almacenados en distintas fuentes de información. La arquitectura OLE DB define consumidores de OLE DB y proveedores de OLE DB. Un consumidor de OLE DB es cualquier sistema o aplicación que utiliza interfaces OLE DB; un proveedor de OLE DB es un componente que expone las interfaces OLE DB.

IBM OLE DB Provider para DB2 permite a DB2 actuar como gestor de recursos para el proveedor de OLE DB. Este soporte ofrece a las aplicaciones basadas en OLE DB la posibilidad de extraer o consultar datos de DB2 mediante la interfaz OLE. IBM OLE DB Provider para DB2, cuyo nombre de proveedor es IBMDADB2, permite a los consumidores de OLE DB acceder a datos en un servidor DB2 Universal Database. Si DB2 Connect está instalado,

estos consumidores de OLE DB también pueden acceder a datos en sistemas principales DBMS como DB2 para MVS, DB2 para VM/VSE o SQL/400.

IBM OLE DB Provider para DB2 ofrece las siguientes funciones:

- Nivel de soporte 0 de la especificación de proveedor de OLE DB, incluidas algunas interfaces adicionales de nivel 1.
- Una implantación del proveedor de hebras libres, que permite a la aplicación crear componentes en una hebra y utilizar dichos componentes en cualquier otra hebra.
- Un Servicio de búsqueda de errores que devuelve mensajes de error de DB2.

Tenga en cuenta que IBM OLE DB Provider reside en el cliente y es distinto de las funciones de tabla de OLE DB, que también reciben soporte de DB2 UDB.

Las siguientes secciones de este documento describen la implantación específica de IBM OLE DB Provider para DB2. Para obtener más información sobre la especificación OLE DB 2.0 de Microsoft, consulte el manual Microsoft OLE DB 2.0 Programmer's Reference and Data Access SDK, disponible en Microsoft Press.

Cumplimiento de versiones:

IBM OLE DB Provider para DB2 cumple con la Versión 2.5 de la especificación OLE DB de Microsoft.

Requisitos del sistema:

Consulte la carta de anuncio correspondiente a IBM OLE DB Provider para DB2 Servers para ver los sistemas operativos Windows soportados.

Para instalar IBM OLE DB Provider para DB2, primero tiene que ejecutar uno de los sistemas operativos soportados mencionados anteriormente. También tiene que instalar DB2 Application Development Client, así como Microsoft Data Access Components (MDAC) Versión 2.7, o superior, que estaba disponible en el momento de escribir este documento en el siguiente sitio: `http://www.microsoft.com/data`.

Consulta relacionada:

- "Soporte de IBM OLE DB Provider de interfaces y componentes de OLE DB" en la página 403

Tipos de aplicaciones soportados por IBM OLE DB Provider para DB2

Con IBM OLE DB Provider para DB2, puede crear los siguientes tipos de aplicaciones:

- Aplicaciones ADO, que incluyen:
  - Aplicaciones Microsoft Visual Studio C++
  - Aplicaciones Microsoft Visual Basic
- Aplicaciones C/C++ que acceden a IBMDADB2 directamente mediante las interfaces OLE DB, incluidas aplicaciones ATL cuyos Objetos de consumidor de acceso a datos se han generado mediante ATL COM AppWizard.

Servicios OLE DB

Las secciones siguientes describen los servicios OLE DB.

Modelo de hebras soportado por IBM OLE DB Provider

IBM OLE DB Provider para DB2 da soporte al modelo de hebras libres, que permite a las aplicaciones crear componentes en una hebra y utilizar dichos componentes en cualquier otra hebra.

Manipulación de objetos grandes con IBM OLE DB Provider

Para obtener y establecer datos como objetos de almacenamiento (DBTYPE_IUNKNOWN) con IBMDADB2, utilice la interfaz ISequentialStream del siguiente modo:

- Para vincular un objeto de almacenamiento a un parámetro, DBOBJECT en la estructura DBBINDING sólo puede contener el valor STGM_READ para el campo dwFlag. IBMDADB2 ejecutará el método Read de la interfaz ISequentialStream del objeto vinculado.
- Para obtener datos de un objeto de almacenamiento, la aplicación debe ejecutar un método Read en la interfaz ISequentialStream del objeto de almacenamiento.
- Cuando se obtienen datos, el valor de la parte de longitud es la longitud de los datos reales, no la longitud del puntero IUnknown.

Conjuntos de filas de esquema soportados por IBM OLE DB Provider

La tabla siguiente muestra los conjuntos de filas de esquema soportados por `IDBSchemaRowset`. Tenga que cuenta que las columnas no soportadas se establecerán en nulo en los conjuntos de filas.

Tabla 20. Conjuntos de filas soportados por IBM OLE DB Provider para DB2

GUID soportados	Restricciones soportadas	Columnas soportadas	Notas
DBSCHEMA _COLUMN_PRIVILEGES	COLUMN_NAME TABLE_NAME TABLE_SCHEMA	COLUMN_NAME GRANTEE GRANTOR IS_GRANTABLE PRIVILEGE_TYPE TABLE_NAME TABLE_SCHEMA	
DB_SCHEMA_COLUMNS	COLUMN_NAME TABLE_NAME TABLE_SCHEMA	CHARACTER_MAXIMUM_LENGTH CHARACTER_OCTET_LENGTH COLUMN_DEFAULT COLUMN_FLAGS COLUMN_HASDEFAULT COLUMN_NAME DATA_TYPE DESCRIPTION IS_NULLABLE NUMERIC_PRECISION NUMERIC_SCALE ORDINAL_POSITION TABLE_NAME TABLE_SCHEMA	
DBSCHEMA_FOREIGN_KEYS	FK_TABLE_NAME FK_TABLE_SCHEMA PK_TABLE_NAME PK_TABLE_SCHEMA	DEFERRABILITY DELETE_RULE FK_COLUMN_NAME FK_NAME FK_TABLE_NAME FK_TABLE_SCHEMA ORDINAL PK_COLUMN_NAME PK_NAME PK_TABLE_NAME PK_TABLE_SCHEMA UPDATE_RULE	Se debe especificar al menos una de las siguientes restricciones: PK_TABLE_NAME o FK_TABLE_NAME No se permiten caracteres comodín “%”.
DBSCHEMA_INDEXES	TABLE_NAME TABLE_SCHEMA	CARDINALITY CLUSTERED COLLATION COLUMN_NAME INDEX_NAME INDEX_SCHEMA ORDINAL_POSITION PAGES TABLE_NAME TABLE_SCHEMA TYPE UNIQUE	No se permite ningún orden de clasificación. El orden de clasificación, si se especifica, se pasará por alto.
DBSCHEMA_PRIMARY_KEYS	TABLE_NAME TABLE_SCHEMA	COLUMN_NAME ORDINAL PK_NAME TABLE_NAME TABLE_SCHEMA	Se debe especificar al menos la siguiente restricción: TABLE_NAME No se permiten caracteres comodín “%”.

Tabla 20. Conjuntos de filas soportados por IBM OLE DB Provider para DB2 (continuación)

GUID soportados	Restricciones soportadas	Columnas soportadas	Notas
DBSCHEMA _PROCEDURE_PARAMETERS	PARAMETER_NAME PROCEDURE_NAME PROCEDURE_SCHEMA	CHARACTER_MAXIMUM_LENGTH CHARACTER_OCTET_LENGTH DATA_TYPE DESCRIPTION IS_NULLABLE NUMERIC_PRECISION NUMERIC_SCALE ORDINAL_POSITION PARAMETER_DEFAULT PARAMETER_HASDEFAULT PARAMETER_NAME PARAMETER_TYPE PROCEDURE_NAME PROCEDURE_SCHEMA TYPE_NAME	
DBSCHEMA_PROCEDURES	PROCEDURE_NAME PROCEDURE_SCHEMA	DESCRIPTION PROCEDURE_NAME PROCEDURE_SCHEMA PROCEDURE_TYPE	
DBSCHEMA_PROVIDER_TYPES	(NONE)	AUTO_UNIQUE_VALUE BEST_MATCH CASE_SENSITIVE CREATE_PARAMS COLUMN_SIZE DATA_TYPE FIXED_PREC_SCALE IS_FIXEDLENGTH IS_LONG IS_NULLABLE LITERAL_PREFIX LITERAL_SUFFIX LOCAL_TYPE_NAME MINIMUM_SCALE MAXIMUM_SCALE SEARCHABLE TYPE_NAME UNSIGNED_ATTRIBUTE	
DBSCHEMA_STATISTICS	TABLE_NAME TABLE_SCHEMA	CARDINALITY TABLE_NAME TABLE_SCHEMA	No se permite ningún orden de clasificación. El orden de clasificación, si se especifica, se pasará por alto.
DBSCHEMA _TABLE_PRIVILEGES	TABLE_NAME TABLE_SCHEMA	GRANTEE GRANTOR IS_GRANTABLE PRIVILEGE_TYPE TABLE_NAME TABLE_SCHEMA	
DBSCHEMA_TABLES	TABLE_NAME TABLE_SCHEMA TABLE_TYPE	DESCRIPTION TABLE_NAME TABLE_SCHEMA TABLE_TYPE	

Habilitación automática de servicios OLE DB por parte de IBM OLE DB Provider

Por omisión, IBM OLE DB Provider para DB2 habilita de forma automática todos los servicios OLE DB, añadiendo una entrada de registro OLEDB_SERVICES bajo el ID de clase (CLSID) del proveedor con el valor 0xFFFFFFFF para DWORD. El significado de este valor es el siguiente:

Tabla 21. Servicios OLE DB

Servicios habilitados	Valor de DWORD
Todos los servicios (valor por omisión)	0xFFFFFFFF
Todos excepto agrupación y AutoEnlistment	0xFFFFFFFE
Todos excepto cursor del cliente	0xFFFFFFFB
Todos excepto agrupación, alistamiento y cursor	0xFFFFFFF0
Ningún servicio	0x000000000

Servicios de datos

Las secciones siguientes describen consideraciones sobre los servicios de datos.

Modalidades de cursor soportadas en IBM OLE DB Provider

IBM OLE DB Provider para DB2 da soporte de forma nativa a cursores de solo lectura y de solo avance, denominados *Cursores del servidor*. Para cursores desplazables actualizables, la aplicación debe utilizar el componente Cursor Service de OLE DB, conocido como Cursor del cliente. Las aplicaciones OLE DB nativas tendrán cursores actualizables y desplazables disponibles cuando se utilicen las interfaces principales de OLE DB `IDataInitialize` o `IDBPromptInitialize` para conectar con la base de datos. Esto se debe a que estas interfaces activan de forma automática el componente Cursor Service de OLE DB.

Correlaciones de tipos de datos entre DB2 y OLE DB

IBM OLE DB Provider da soporte a las correlaciones de tipos de datos entre tipos de datos DB2 y tipos de datos OLE DB. La tabla siguiente proporciona una lista completa de correlaciones soportadas y nombres disponibles para indicar los tipos de datos de columnas y parámetros.

Tabla 22. Correlaciones de tipos de datos entre tipos de datos DB2 y tipos de datos OLE DB

Tipos de datos DB2	**Indicadores de tipos de datos OLE DB**	**Nombres de tipos estándar de OLE DB**	**Nombres específicos de DB2**
SMALLINT	DBTYPE_I2	“DBTYPE_I2”	“SMALLINT”
INTEGER	DBTYPE_I4	“DBTYPE_I4”	“INTEGER” o “INT”
BIGINT	DBTYPE_I8	“DBTYPE_I8”	“BIGINT”
REAL	DBTYPE_R4	“DBTYPE_R4”	“REAL”
FLOAT	DBTYPE_R8	“DBTYPE_R8”	“FLOAT”
DOUBLE	DBTYPE_R8	“DBTYPE_R8″	“DOUBLE” o “DOUBLE PRECISION”
DECIMAL	DBTYPE_NUMERIC	“DBTYPE_NUMERIC”	“DEC” o “DECIMAL”
NUMERIC	DBTYPE_NUMERIC	“DBTYPE_NUMERIC”	“NUM” o “NUMERIC”
DATE	DBTYPE_DBDATE	“DBTYPE_DBDATE”	“DATE”
TIME	DBTYPE_DBTIME	“DBTYPE_DBTIME”	“TIME”
TIMESTAMP	DBTYPE_DBTIMESTAMP	“DBTYPE_DBTIMESTAMP”	“TIMESTAMP”
CHAR	DBTYPE_STR	“DBTYPE_CHAR”	“CHAR” o “CHARACTER”
VARCHAR	DBTYPE_STR	“DBTYPE_VARCHAR”	“VARCHAR”
LONG VARCHAR	DBTYPE_STR	“DBTYPE_LONGVARCHAR”	“LONG VARCHAR”
CLOB	DBTYPE_STR y DBCOLUMNFLAGS_ISLONG o DBPARAMFLAGS_ISLONG	“DBTYPE_CHAR” “DBTYPE_VARCHAR” “DBTYPE_LONGVARCHAR” y DBCOLUMNFLAGS_ISLONG o DBPARAMFLAGS_ISLONG	“CLOB”
GRAPHIC	DBTYPE_WSTR	“DBTYPE_WCHAR”	“GRAPHIC”
VARGRAPHIC	DBTYPE_WSTR	“DBTYPE_WVARCHAR”	“VARGRAPHIC”
LONG VARGRAPHIC	DBTYPE_WSTR	“DBTYPE_WLONGVARCHAR”	“LONG VARGRAPHIC”
DBCLOB	DBTYPE_WSTR y DBCOLUMNFLAGS_ISLONG o DBPARAMFLAGS_ISLONG	“DBTYPE_WCHAR” “DBTYPE_WVARCHAR” “DBTYPE_WLONGVARCHAR” y DBCOLUMNFLAGS_ISLONG o DBPARAMFLAGS_ISLONG	“DBCLOB”
CHAR(n) FOR BIT DATA	DBTYPE_BYTES	“DBTYPE_BINARY”	
VARCHAR(n) FOR BIT DATA	DBTYPE_BYTES	“DBTYPE_VARBINARY”	
LONG VARCHAR FOR BIT DATA	DBTYPE_BYTES	“DBTYPE_LONGVARBINARY”	

Tabla 22. Correlaciones de tipos de datos entre tipos de datos DB2 y tipos de datos OLE DB (continuación)

Tipos de datos DB2	Indicadores de tipos de datos OLE DB	Nombres de tipos estándar de OLE DB	Nombres específicos de DB2
BLOB	DBTYPE_BYTES y DBCOLUMNFLAGS_ISLONG o DBPARAMFLAGS_ISLONG	"DBTYPE_BINARY" "DBTYPE_VARBINARY" "DBTYPE_LONGVARBINARY" y DBCOLUMNFLAGS_ISLONG o DBPARAMFLAGS_ISLONG	"BLOB"
DATA LINK	DBTYPE_STR	"DBTYPE_CHAR"	"DATA LINK"

Conversión de datos para establecer datos de tipos OLE DB en tipos DB2

IBM OLE DB Provider da soporte a las conversiones de datos para establecer datos de tipos OLE DB en tipos DB2. Tenga en cuenta que es posible que se trunquen datos en algunos casos, en función de los tipos y del valor de los datos.

Tabla 23. Conversión de datos de tipos OLE DB a tipos DB2

	Tipos de datos DB2																					
																			For Bit Data			
Indicador de tipo OLE DB	SMALLINT	INTEGER	BIGINT	REAL	FLOAT DOUBLE	DECIMAL NUMERIC	DATE	TIME	TIMESTAMP	CHAR	VARCHAR	LONG VARCHAR	CLOB	GRAPHIC	VARGRAPHIC	LONG VARGRAPHIC	DBCLOB	CHAR	VARCHAR	LONG VARCHAR	BLOB	DATALINK
DBTYPE_EMPTY																						
DBTYPE_NULL																						
DBTYPE_RESERVED																						
DBTYPE_I1	X	X	X	X	X	X				X	X											
DBTYPE_I2	X	X	X	X	X	X				X	X											
DBTYPE_I4	X	X	X	X	X	X				X	X											
DBTYPE_I8	X	X	X	X	X	X				X	X											
DBTYPE_UI1	X	X	X	X	X	X				X	X											
DBTYPE_UI2	X	X	X	X	X	X				X	X											
DBTYPE_UI4	X	X	X	X	X	X				X	X											
DBTYPE_UI8	X	X	X	X	X	X				X	X											

Tabla 23. Conversión de datos de tipos OLE DB a tipos DB2 (continuación)

	Tipos de datos DB2																					
																			For Bit Data			
Indicador de tipo OLE DB	**SMALLINT**	**INTEGER**	**BIGINT**	**REAL**	**FLOAT DOUBLE**	**DECIMAL NUMERIC**	**DATE**	**TIME**	**TIMESTAMP**	**CHAR**	**VARCHAR**	**LONG VARCHAR**	**CLOB**	**GRAPHIC**	**VARGRAPHIC**	**LONG VARGRAPHIC**	**DBCLOB**	**CHAR**	**VARCHAR**	**LONG VARCHAR**	**BLOB**	**DATALINK**
DBTYPE_R4	X	X	X	X	X	X				X	X											
DBTYPE_R8	X	X	X	X	X	X				X	X											
DBTYPE_CY																						
DBTYPE_DECIMAL	X	X	X	X	X	X				X	X											
DBTYPE_NUMERIC	X	X	X	X	X	X				X	X											
DBTYPE_DATE																						
DBTYPE_BOOL	X	X	X	X	X	X				X	X											
DBTYPE_BYTES			X			X				X	X	X				X		X	X	X		
DBTYPE_BSTR – por determinar																						
DBTYPE_STR	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_WSTR														X	X	X						
DBTYPE_VARIANT – por determinar																						
DBTYPE_IDISPATCH																						
DBTYPE_IUNKNOWN										X	X	X	X	X	X	X	X	X	X	X		
DBTYPE_GUID																						
DBTYPE_ERROR																						
DBTYPE_BYREF																						
DBTYPE_ARRAY																						
DBTYPE_VECTOR																						
DBTYPE_UDT																						
DBTYPE_DBDATE							X		X	X	X											
DBTYPE_DBTIME								X	X	X	X											
DBTYPE_DBTIMESTAMP							X	X	X	X	X											
DBTYPE_FILETIME																						

Tabla 23. Conversión de datos de tipos OLE DB a tipos DB2 (continuación)

	Tipos de datos DB2																					
																			For Bit Data			
Indicador de tipo OLE DB	**SMALLINT**	**INTEGER**	**BIGINT**	**REAL**	**FLOAT DOUBLE**	**DECIMAL NUMERIC**	**DATE**	**TIME**	**TIMESTAMP**	**CHAR**	**VARCHAR**	**LONG VARCHAR**	**CLOB**	**GRAPHIC**	**VARGRAPHIC**	**LONG VARGRAPHIC**	**DBCLOB**	**CHAR**	**VARCHAR**	**LONG VARCHAR**	**BLOB**	**DATALINK**
DBTYPE_PROP_VARIANT																						
DBTYPE_HCHAPTER																						
DBTYPE_VARNUMERIC																						

Consulta relacionada:

- "Conversión de datos para establecer datos de tipos DB2 en tipos OLE DB" en la página 400

Conversión de datos para establecer datos de tipos DB2 en tipos OLE DB

Para obtener datos, IBM OLE DB Provider permite conversiones de datos de tipos DB2 a tipos OLE DB. Tenga en cuenta que es posible que se trunquen datos en algunos casos, en función de los tipos y del valor de los datos.

Tabla 24. Conversiones de datos de tipos DB2 a tipos OLE DB

	Tipos de datos DB2																				
																		For Bit Data			
Indicador de tipo OLE DB	SMALLINT	INTEGER	BIGINT	REAL	FLOAT DOUBLE	DECIMAL NUMERIC	DATE	TIME	TIMESTAMP	CHAR	VARCHAR	LONG VARCHAR	CLOB	GRAPHIC	VARGRAPHIC	LONG VARGRAPHIC	DBCLOB	CHAR	VARCHAR	LONG VARCHAR	BLOB	DATALINK
DBTYPE_EMPTY																					
DBTYPE_NULL																					
DBTYPE_RESERVED																					
DBTYPE_I1	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_I2	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_I4	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_I8	X	X	X	X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_UI1	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_UI2	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_UI4	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_UI8	X	X	X	X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_R4	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_R8	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_CY	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_DECIMAL	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_NUMERIC	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_DATE	X	X		X	X		X	X	X	X	X	X		X	X	X						X
DBTYPE_BOOL	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_BYTES	X	X		X	X	X	X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_BSTR	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_STR	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_WSTR	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_VARIANT	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_IDISPATCH																						
DBTYPE_IUNKNOWN	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
DBTYPE_GUID										X	X	X		X	X	X		X	X	X		X

Tabla 24. Conversiones de datos de tipos DB2 a tipos OLE DB (continuación)

	Tipos de datos DB2																					
																			For Bit Data			
Indicador de tipo OLE DB	**SMALLINT**	**INTEGER**	**BIGINT**	**REAL**	**FLOAT DOUBLE**	**DECIMAL NUMERIC**	**DATE**	**TIME**	**TIMESTAMP**	**CHAR**	**VARCHAR**	**LONG VARCHAR**	**CLOB**	**GRAPHIC**	**VARGRAPHIC**	**LONG VARGRAPHIC**	**DBCLOB**	**CHAR**	**VARCHAR**	**LONG VARCHAR**	**BLOB**	**DATALINK**
DBTYPE_ERROR																						
DBTYPE_BYREF																						
DBTYPE_ARRAY																						
DBTYPE_VECTOR																						
DBTYPE_UDT																						
DBTYPE_DBDATE							X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_DBTIME							X	X	X	X	X	X		X	X	X						X
DBTYPE_DBTIMESTAMP							X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_FILETIME			X				X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_PROP_VARIANT	X	X	X	X	X					X	X	X		X	X	X		X	X	X		X
DBTYPE_HCHAPTER																						
DBTYPE_VARNUMERIC																						

Nota: cuando la aplicación realiza una operación ISequentialStream::Read para obtener los datos del objeto de almacenamiento, el formato de los datos devueltos depende del tipo de datos de la columna:

- Para tipos de datos binarios y que no sean de tipo carácter, los datos de la columna se exponen como una secuencia de bytes que representan dichos valores en el sistema operativo.
- Para tipos de datos de carácter, primero los datos se convierten a DBTYPE_STR.
- Para DBCLOB, primero los datos se convierten a DBTYPE_WCHAR.

Consulta relacionada:

- "Conversión de datos para establecer datos de tipos OLE DB en tipos DB2" en la página 398

Restricciones de IBM OLE DB Provider

A continuación se describen las restricciones de IBM OLE DB Provider:

- IBMDADB2 da soporte al ámbito de transacciones de confirmación automática y controladas por el usuario con la interfaz `ITransactionLocal`. El ámbito de transacciones de confirmación automática es el ámbito por omisión. Las transacciones anidadas no reciben soporte.
- `ISQLErrorInfo` no recibe soporte. Las interfaces `IErrorLookUp`, `IErrorInfo` y `IErrorRecords` reciben soporte.
- `RestartPosition` no recibe soporte si el texto del mandato contiene parámetros.
- IBMDADB2 no pone entre comillas nombres de tablas que se pasan a través de parámetros de DBID, que son los parámetros que utiliza la interfaz `IOpenRowset`. En su lugar, el consumidor de OLE DB debe añadir comillas a los nombres de tablas cuando sea necesario.
- Sólo se da soporte a un solo conjunto de parámetros. Todavía no se da soporte a varios conjuntos de parámetros.
- IBM OLE DB Provider no da soporte a parámetros con nombre. Cuando se llama a `ICommandWithParameters::MapParameterNames`, siempre se devuelve DB_S_ERRORSOCCURRED. Los nombres de parámetros se pasan por alto en `ICommandWithParameters::GetParameterInfo` y `ICommandWithParameters::SetParameterInfo`, puesto que sólo se utilizan ordinales.

Soporte de IBM OLE DB Provider de interfaces y componentes de OLE DB

La tabla siguiente contiene las interfaces y componentes de OLE DB que reciben soporte de IBM OLE DB Provider y de Microsoft OLE DB Provider para ODBC.

Tabla 25. Comparación de interfaces y componentes de OLE DB soportados por IBM OLE DB Provider para DB2 y Microsoft OLE DB Provider para ODBC

	Interfaz	**DB2**	**ODBC Provider**
BLOB			
	ISequentialStream	Sí	Sí
Mandato			
	IAccessor	Sí	Sí
	ICommand	Sí	Sí
	ICommandPersist	No	No
	ICommandPrepare	Sí	Sí
	ICommandProperties	Sí	Sí
	ICommandText	Sí	Sí
	ICommandWithParameters	Sí	Sí

Tabla 25. Comparación de interfaces y componentes de OLE DB soportados por IBM OLE DB Provider para DB2 y Microsoft OLE DB Provider para ODBC (continuación)

	Interfaz	**DB2**	**ODBC Provider**
	IColumnsInfo	Sí	Sí
	IColumnsRowset	No	Sí
	IConvertType	Sí	Sí
	ISupportErrorInfo	Sí	Sí
Fuente de datos			
	IConnectionPoint	No	Sí
	IDBAsynchNotify (consumer)	No	No
	IDBAsynchStatus	No	No
	IDBConnectionPointContainer	No	Sí
	IDBCreateSession	Sí	Sí
	IDBDataSourceAdmin	No	No
	IDBInfo	Sí	Sí
	IDBInitialize	Sí	Sí
	IDBProperties	Sí	Sí
	IPersist	Sí	No
	IPersistFile	No	Sí
	ISupportErrorInfo	Sí	Sí
Enumerador			
	IDBInitialize	Sí	Sí
	IDBProperties	Sí	Sí
	IParseDisplayName	Sí	No
	ISourcesRowset	Sí	Sí
	ISupportErrorInfo	Sí	Sí
Servicio de búsqueda de errores			
	IErrorLookUp	Sí	Sí
Objeto Error			
	IErrorInfo	Sí	No
	IErrorRecords	Sí	No
	ISQLErrorInfo (custom)	No	No
Varios resultados			
	IMultipleResults	Sí	Sí

Tabla 25. Comparación de interfaces y componentes de OLE DB soportados por IBM OLE DB Provider para DB2 y Microsoft OLE DB Provider para ODBC (continuación)

	Interfaz	**DB2**	**ODBC Provider**
	ISupportErrorInfo	Sí	Sí
Conjunto de filas			
	IAccessor	Sí	Sí
	IColumnsRowset	No	Sí
	IColumnsInfo	Sí	Sí
	IConvertType	Sí	Sí
	IChapteredRowset	No	No
	IConnectionPointContainer	No	Sí
	IDBAsynchStatus	No	No
	IParentRowset	No	No
	IRowset	Sí	Sí
	IRowsetChange	Componente Servicio de cursor	Sí
	IRowsetChapterMember	No	No
	IRowsetFind	No	No
	IRowsetIdentity	Sí	Sí
	IRowsetIndex	No	No
	IRowsetInfo	Sí	Sí
	IRowsetLocate	Componente Servicio de cursor	Sí
	IRowsetNotify (consumer)	No	No
	IRowsetRefresh	Componente Servicio de cursor	Sí
	IRowsetResynch	Componente Servicio de cursor	Sí
	IRowsetScroll	Componente Servicio de cursor	Sí
	IRowsetUpdate	Componente Servicio de cursor	Sí
	IRowsetView	No	No
	ISupportErrorInfo	Sí	Sí
Sesiones			
	IAlterIndex	No	No

Tabla 25. Comparación de interfaces y componentes de OLE DB soportados por IBM OLE DB Provider para DB2 y Microsoft OLE DB Provider para ODBC (continuación)

	Interfaz	**DB2**	**ODBC Provider**
	IAlterTable	No	No
	IDBCreateCommand	Sí	Sí
	IDBSchemaRowset	Sí	Sí
	IGetDataSource	Sí	Sí
	IIndexDefinition	No	No
	IOpenRowset	Sí	Sí
	ISessionProperties	Sí	Sí
	ISupportErrorInfo	Sí	Sí
	ITableDefinition	No	No
	ITableDefinitionWithConstraints	No	No
	ITransaction	Sí	Sí
	ITransactionJoin	Sí	Sí
	ITransactionLocal	Sí	Sí
	ITransactionObject	No	No
	ITransactionOptions	No	Sí
Objetos Vista			
	IViewChapter	No	No
	IViewFilter	No	No
	IViewRowset	No	No
	IViewSort	No	No

Soporte de IBM OLE DB de propiedades de OLE DB

La tabla siguiente muestra las propiedades de OLE DB que reciben soporte de IBM OLE DB Provider:

Tabla 26. Propiedades soportadas por IBM OLE DB Provider para DB2

Grupo de propiedades	**Conjunto de propiedades**	**Propiedades**	**Valor por omisión**	**R/W**
Fuente de datos	DBPROPSET_DATASOURCE	DBPROP_MULTIPLECONNECTIONS	VARIANT_FALSE	R
		DBPROP_RESETDATASOURCE	DBPROPVAL_RD_RESETALL	R
Información de fuente de datos	DBPROPSET _DATASOURCEINFO	DBPROP_ACTIVESESSIONS	0	R
		DBPROP_ASYNCTXNABORT	VARIANT_FALSE	R

Tabla 26. Propiedades soportadas por IBM OLE DB Provider para DB2 (continuación)

Grupo de propiedades	Conjunto de propiedades	Propiedades	Valor por omisión	R/W
		DBPROP_ASYNCTXNCOMMIT	VARIANT_FALSE	R
		DBPROP_BYREFACCESSORS	VARIANT_FALSE	R
		DBPROP_COLUMNDEFINITION	DBPROPVAL_CD_NOTNULL	R
		DBPROP_CONCATNULLBEHAVIOR	DBPROPVAL_CB_NULL	R
		DBPROP_CONNECTIONSTATUS	DBPROPVAL_CS_INITIALIZED	R
		DBPROP_DATASOURCENAME	N/D	R
		DBPROP_DATASOURCEREADONLY	VARIANT_FALSE	R
		DBPROP_DBMSNAME	N/D	R
		DBPROP_DBMSVER	N/D	R
		DBPROP_DSOTHREADMODEL	DBPROPVAL_RT_FREETHREAD	R
		DBPROP_GROUPBY	DBPROPVAL_GB_CONTAINS_SELECT	R
		DBPROP_IDENTIFIERCASE	DBPROPVAL_IC_UPPER	R
		DBPROP_MAXINDEXSIZE	0	R
		DBPROP_MAXROWSIZE	0	R
		DBPROP_MAXROWSIZEINCLUDESBLOB	VARIANT_TRUE	R
		DBPROP_MAXTABLEINSELECT	0	R
		DBPROP_MULTIPLEPARAMSETS	VARIANT_FALSE	R
		DBPROP_MULTIPLERESULTS	DBPROPVAL_MR_SUPPORTED	R
		DBPROP_MULTIPLESTORAGEOBJECTS	VARIANT_TRUE	R
		DBPROP_MULTITABLEUPDATE	VARIANT_FALSE	R
		DBPROP_NULLCOLLATION	DBPROPVAL_NC_LOW	R
		DBPROP_OLEOBJECTS	DBPROPVAL_OO_BLOB	R
		DBPROP_ORDERBYCOLUMNSINSELECT	VARIANT_FALSE	R
		DBPROP _OUTPUTPARAMETERAVAILABILITY	DBPROPVAL_OA_ATEXECUTE	R
		DBPROP_PERSISTENTIDTYPE	DBPROPVAL_PT_NAME	R
		DBPROP_PREPAREABORTBEHAVIOR	DBPROPVAL_CB_DELETE	R
		DBPROP_PROCEDURETERM	“PROCEDIMIENTO ALMACENADO”	R
		DBPROP_PROVIDERFRIENDLYNAME	“Servidores IBM OLE DB Provider para DB2”	R
		DBPROP_PROVIDERNAME	“IBMDADB2.DLL”	R
		DBPROP_PROVIDEROLEDBVER	“02.00”	R
		DBPROP_PROVIDERVER	“08.01.0000”	R
		DBPROP_QUOTEIDENTIFIERCASE	DBPROPVAL_IC_SENSITIVE	R
		DBPROP _ROWSETCONVERSIONSONCOMMAND	VARIANT_TRUE	R
		DBPROP_SCHEMATERM	“ESQUEMA”	R
		DBPROP_SCHEMAUSAGE	DBPROPVAL_SU_DML_STATEMENTS | DBPROPVAL_SU_TABLE_DEFINITION | DBPROPVAL_SU_INDEX_DEFINITION | DBPROPVAL_SU_PRIVILEGE_DEFINITION	R
		DBPROP_SQLSUPPORT	DBPROPVAL_SQL_ODBC_EXTENDED | DBPROPVAL_SQL_ESCAPECLAUSES | DBPROPVAL_SQL_ANSI92_ENTRY	R
		DBPROP_SERVERNAME	N/D	R
		DBPROP_STRUCTUREDSTORAGE	DBPROPVAL_SS_ISEQUENTIALSTREAM	R
		DBPROP_SUBQUERIES	DBPROPVAL_SQ_CORRELATEDSUBQUERIES DBPROPVAL_SQ_COMPARISON | DBPROPVAL_SQ_EXISTS | DBPROPVAL_SQ_IN | DBPROPVAL_SQ_QUANTIFIED |	R

Tabla 26. Propiedades soportadas por IBM OLE DB Provider para DB2 (continuación)

Grupo de propiedades	Conjunto de propiedades	Propiedades	Valor por omisión	R/W
		DBPROP_SUPPORTEDTXNDDL	DBPROPVAL_TC_ALL	R
		DBPROP_SUPPORTEDTXNISOLEVELS	DBPROPVAL_TI_CURSORSTABILITY | DBPROPVAL_TI_READCOMMITTED | DBPROPVAL_TI_READUNCOMMITTED | DBPROPVAL_TI_SERIALIZABLE |	R
		DBPROP_SUPPORTEDTXNISORETAIN	DBPROPVAL_TR_COMMIT_DC | DBPROPVAL_TR_ABORT_NO |	R
		DBPROP_TABLETERM	"TABLA"	R
		DBPROP_USERNAME	N/D	R
Inicialización	DBPROPSET_DBINIT	DBPROP_AUTH_PASSWORD	N/D	R/W
		DBPROP_AUTH_USERID	N/D	R/W
		DBPROP_INIT_DATASOURCE	N/D	R/W
		DBPROP_INIT_HWND	N/D	R/W
		DBPROP_INIT_MODE	DB_MODE_READWRITE	R/W
		DBPROP_INIT_OLEDBSERVICES	0xFFFFFFFF	R/W
		DBPROP_INIT_PROMPT	DBPROMPT_NOPROMPT	R/W
		DBPROP_INIT_PROVIDERSTRING	N/D	R/W
Conjunto de filas	DBPROPSET_ROWSET	DBPROP_ABORTPRESERVE	VARIANT_FALSE	R
		DBPROP_ACCESSORDER	DBPROPVAL_AO_RANDOM	R
		DBPROP_BLOCKINGSTORAGEOBJECTS	VARIANT_FALSE	R
		DBPROP_CACHEDEFERRED	VARIANT_FALSE	R/W
		DBPROP_CANHOLDROWS	VARIANT_FALSE	R
		DBPROP_COMMITPRESERVE	VARIANT_TRUE	R/W
		DBPROP_COMMANDTIMEOUT	0	R/W
		DBPROP_DEFERRED	VARIANT_FALSE	R
		DBPROP_IAccessor	VARIANT_TRUE	R
		DBPROP_IColumnsInfo	VARIANT_TRUE	R
		DBPROP_IColumnsRowset	VARIANT_TRUE	R
		DBPROP_IConvertType	VARIANT_TRUE	R
		DBPROP_IMultipleResults	VARIANT_TRUE	R
		DBPROP_IRowset	VARIANT_TRUE	R
		DBPROP_IRowChange	VARIANT_FALSE	R
		DBPROP_IRowsetFind	VARIANT_FALSE	R
		DBPROP_IRowsetIdentity	VARIANT_TRUE	R
		DBPROP_IRowsetInfo	VARIANT_TRUE	R
		DBPROP_IRowsetLocate	VARIANT_FALSE	R
		DBPROP_IRowsetScroll	VARIANT_FALSE	R
		DBPROP_IRowsetUpdate	VARIANT_FALSE	R
		DBPROP_ISequentialStream	VARIANT_TRUE	R
		DBPROP_ISupportErrorInfo	VARIANT_TRUE	R
		DBPROP_LITERALIDENTITY	VARIANT_TRUE	R
		DBPROP_LOCKMODE	DBPROPVAL_LM_SINGLEROW	R/W
		DBPROP_MAXOPENROWS	0	R/W
		DBPROP_MAXROWS	0	R/W
		DBPROP_QUICKRESTART	VARIANT_FALSE	R/W
		DBPROP_ROWTHREADMODEL	DBPROPVAL_RT_FREETHREAD	R
		DBPROP_SERVERCURSOR	VARIANT_TRUE	R

Tabla 26. Propiedades soportadas por IBM OLE DB Provider para DB2 (continuación)

Grupo de propiedades	Conjunto de propiedades	Propiedades	Valor por omisión	R/W
		DBPROP_UNIQUEROWS	VARIANT_FALSE	R
	DBPROPSET_DB2ROWSET	DBPROP_OPENROWSETSUPPORT	DBPROPVAL_ORS_TABLE	R
		DBPROP_ISLONGMINLENGTH	32000	R/W
Sesión	DBPROPSET_SESSION	DBPROP_SESS_AUTOCOMMITISOLEVELS	DBPROPVAL_TI_CURSORSTABILITY	R/W

Conexiones a fuentes de datos mediante IBM OLE DB Provider

Los siguientes ejemplos muestran cómo conectar con una fuente de datos DB2 mediante IBM OLE DB Provider para DB2:

Ejemplo 1: Aplicación Visual Basic que utiliza ADO:

```
Dim db As ADODB.Connection
Set db = New ADODB.Connection
db.Provider = "IBMDADB2"
db.CursorLocation = adUseClient
...
```

Ejemplo 2: Aplicación C/C++ que utiliza IDBPromptInitialize y Data Links:

```
// Crear DataLinks
hr = CoCreateInstance (
     CLSID_DataLinks,
     NULL,
     CLSCTX_INPROC_SERVER,
     IID_IDBPromptInitialize,
     (void**)&pIDBPromptInitialize);

// Invocar la UI DataLinks para seleccionar el proveedor y la fuente de datos
hr = pIDBPromptInitialize->PromptDataSource (
     NULL,
     GetDesktopWindow(),
     DBPROMPTOPTIONS_PROPERTYSHEET,
     0,
     NULL,
     NULL,
     IID_IDBInitialize,
     (IUnknown**)&pIDBInitialize);
```

Ejemplo 3: Aplicación C/C++ que utiliza IDataInitialize y el componente de servicio:

```
hr = CoCreateInstance (
     CLSID_MSDAINITIALIZE,
     NULL,
     CLSCTX_INPROC_SERVER,
     IID_IDataInitialize,
     (void**)&pIDataInitialize);
```

```
hr = pIDataInitialize->CreateDBInstance(
     CLSID_IBMDADB2, // ClassID de IBMDADB2
     NULL,
     CLSCTX_INPROC_SERVER,
     NULL,
     IID_IDBInitialize,
     (IUnknown**)&pIDBInitialize);
```

Aplicaciones ADO

Las siguientes secciones describen consideraciones sobre aplicaciones ADO.

Palabras clave de series de conexión de ADO

Para especificar palabras clave de series de conexión de ADO (Objetos de datos ActiveX), especifique la palabra clave con el formato `palabra clave=`*`valor`* en la serie (conexión) del proveedor. Delimite varias palabras clave con un punto y coma (;).

La tabla siguiente describe las palabras clave a las que da soporte IBM OLE DB Provider para DB2:

Tabla 27. Palabras clave a las que da soporte IBM OLE DB Provider para DB2

Palabra clave	Valor	Significado
DSN	Nombre del alias de la base de datos	El alias de la base de datos DB2 en el directorio de bases de datos.
UID	ID de usuario	El ID de usuario que se utiliza para conectar con el servidor DB2.
PWD	Contraseña de UID	Contraseña correspondiente al ID de usuario utilizado para conectar con el servidor DB2.

Hay otras palabras clave de configuración de CLI de DB2 que también afectan al comportamiento de IBM OLE DB Provider.

Consulta relacionada:

- "CLI/ODBC Configuration Keywords Listing by Category" en el manual *CLI Guide and Reference, Volume 1*

Conexiones con fuentes de datos con aplicaciones ADO Visual Basic

Para conectar con una fuente de datos DB2 mediante IBM OLE DB Provider para DB2, especifique el nombre de proveedor IBMDADB2.

Conceptos relacionados:

- "Conexiones a fuentes de datos mediante IBM OLE DB Provider" en la página 409

Tareas relacionadas:

- "Creación de aplicaciones ADO con Visual Basic" en el manual *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

Cursores desplazables actualizables en aplicaciones ADO

Puesto que IBM OLE DB Provider para DB2 da soporte de forma nativa a cursores de solo lectura y de solo avance, una aplicación ADO que desee acceder a cursores desplazables actualizables debe establecer como ubicación del cursor el valor `adUseClient`.

Limitaciones de las aplicaciones ADO

A continuación se describen las limitaciones de las aplicaciones ADO:

- Las aplicaciones ADO que llaman a procedimientos almacenados deben crear y vincular de forma explícita sus parámetros. El método `Parameters.Refresh` para generar parámetros automáticamente aún no recibe soporte.
- No se da soporte a valores por omisión de parámetros.
- Para aplicaciones ADO Visual Basic, los controles de datos no reciben soporte para cursores del servidor. Sin embargo, hay cursores de datos disponibles para aplicaciones de cursor del cliente.
- La palabra clave `WithEvents` no se puede utilizar en declaraciones del objeto de conjunto de registros para aplicaciones ADO Visual Basic que utilizan el cursor del servidor de solo lectura / solo avance; es decir, cuando se especifica para `Cursor Location` el valor `adUseServer`.

Soporte de IBM OLE DB Provider de propiedades y métodos ADO

IBM OLE DB Provider da soporte a los siguientes métodos y propiedades ADO:

Tabla 28. Métodos y propiedades ADO soportados por IBM OLE DB Provider para DB2

ADO	Método/Propiedad	Interfaz/Propiedad OLE DB	Soporte de IBM OLE DB
Métodos de mandato	Cancel	ICommand	Sí
	CreateParameter		Sí
	Execute		Sí
Propiedades de mandato	ActiveConnection	(Específico de ADO)	

Tabla 28. Métodos y propiedades ADO soportados por IBM OLE DB Provider para DB2 (continuación)

ADO	Método/Propiedad	Interfaz/Propiedad OLE DB	Soporte de IBM OLE DB
	Command Text	ICommandText	Sí
	Command Timeout	ICommandProperties::SetProperties DBPROP_COMMANDTIMEOUT	Sí
	CommandType	(Específico de ADO)	
	Prepared	ICommandPrepare	Sí
	State	(Específico de ADO)	
Grupo de mandatos	Parameters	ICommandWithParameter DBSCHEMA _PROCEDURE_PARAMETERS	Sí
	Properties	ICommandProperties IDBProperties	Sí
Métodos de conexión	BeginTrans CommitTrans RollbackTrans	ITransactionLocal	Sí (pero no anidado) Sí (pero no anidado) Sí (pero no anidado)
	Execute	ICommand IOpenRowset	Sí
	Open	IDBCreateSession IDBInitialize	Sí
	OpenSchema adSchemaColumnPrivileges adSchemaColumns adSchemaForeignKeys adSchemaIndexes adSchemaPrimaryKeys adSchemaProcedureParam adSchemaProcedures adSchemaProviderType adSchemaStatistics adSchemaTablePrivileges adSchemaTables	IDBSchemaRowset	 Sí Sí Sí Sí Sí Sí Sí Sí Sí Sí Sí
	Cancel		Sí
Propiedades de conexión	Attributes adXactCommitRetaining adXactRollbackRetaining	ITransactionLocal	 Sí Sí
	CommandTimeout	ICommandProperties DBPROP_COMMAND_TIMEOUT	Sí
	ConnectionString	(Específico de ADO)	
	ConnectionTimeout	IDBProperties DBPROP_INIT_TIMEOUT	No
	CursorLocation: adUseClient adUseNone adUseServer	 (Utilizar Servicio de cursor de OLE DB) (No utilizado) (No actualizable sólo en avance)	 Sí NoSí

Tabla 28. Métodos y propiedades ADO soportados por IBM OLE DB Provider para DB2 (continuación)

ADO	Método/Propiedad	Interfaz/Propiedad OLE DB	Soporte de IBM OLE DB
	DefaultDataBase	IDBProperties DBPROP_CURRENTCATALOG	No
	IsolationLevel	ITransactionLocal DBPROP_SESS _AUTOCOMMITSOLEVELS	Sí
	Mode adModeRead adModeReadWrite adModeShareDenyNone adModeShareDenyRead adModeShareDenyWrite adModeShareExclusive adModeUnknown adModeWrite	IDBProperties DBPROP_INIT_MODE	 NoSí NoNoNoNoNoNo
	Provider	ISourceRowset::GetSourceRowset	Sí
	State	(Específico de ADO)	
	Version	(Específico de ADO)	
Grupos de conexiones	Errors	IErrorRecords	Sí
	Properties	IDBProperties	Sí
Propiedades de error	Description NativeError Number Source SQLState	IErrorRecords	Sí Sí Sí Sí No
	HelpContext HelpFile		No No
Métodos de campo	AppendChunk GetChunk	ISequentialStream	Sí Sí
Propiedades de campo	Actual Size	IAccessor IRowset	Sí
	Attributes DataFormat DefinedSize Name NumericScale Precision Type	IColumnInfo	 Sí Sí Sí Sí Sí Sí
	OriginalValue	IRowsetUpdate	Sí (Servicio de cursor)
	UnderlyingValue	IRowsetRefresh IRowsetResynch	Sí (Servicio de cursor) Sí (Servicio de cursor)

Tabla 28. Métodos y propiedades ADO soportados por IBM OLE DB Provider para DB2 (continuación)

ADO	Método/Propiedad	Interfaz/Propiedad OLE DB	Soporte de IBM OLE DB
	Value	IAccessor IRowset	Sí
Grupo de campos	Properties	IDBProperties IRowsetInfo	Sí
Métodos de parámetro	AppendChunk	ISequentialStream	Sí
	Attributes Direction Name NumericScale Precision Scale Size Type	ICommandWithParameter DBSCHEMA _PROCEDURE_PARAMETERS	 Sí No Sí Sí Sí Sí Sí
	Value	IAccessor ICommand	Sí
Grupo de parámetros	Properties		Sí
Métodos RecordSet	AddNew	IRowsetChange	Sí (Servicio de cursor)
	Cancel		Sí
	CancelBatch	IRowsetUpdate::Undo	Sí (Servicio de cursor)
	CancelUpdate		Sí (Servicio de cursor)
	Clone	IRowsetLocate	Sí (Servicio de cursor)
	Close	IAccessor IRowset	Sí
	CompareBookmarks		No
	Delete	IRowsetChange	Sí (Servicio de cursor)
	GetRows	IAccessor IRowset	Sí (Servicio de cursor)
	Move	IRowset IRowsetLocate	Cursor de servidor: sólo en avance Servicio de cursor: desplazable
	MoveFirst	IRowset IRowsetLocate	Sí (Servicio de cursor)
	MoveNext	IRowset IRowsetLocate	Sí (Servicio de cursor)
	MoveLast	IRowsetLocate	Sí (Servicio de cursor)
	MovePrevious	IRowsetLocate	Sí (Servicio de cursor)
	NextRecordSet	IMultipleResults	Sí

Tabla 28. Métodos y propiedades ADO soportados por IBM OLE DB Provider para DB2 (continuación)

ADO	Método/Propiedad	Interfaz/Propiedad OLE DB	Soporte de IBM OLE DB
	Open	ICommand IOpenRowset	Sí
	Requery	ICommand IOpenRowset	Sí
	Resync	IRowsetRefresh	Sí (Servicio de cursor)
	Supports	IRowsetInfo	Sí
	Update UpdateBatch	IRowsetChange IRowsetUpdate	Sí (Servicio de cursor) Sí (Servicio de cursor)
Propiedades de RecordSet	AbsolutePage	IRowsetLocate IRowsetScroll	Sí (Servicio de cursor)
	AbsolutePosition	IRowsetLocate IRowsetScroll	Sí (Servicio de cursor)
	ActiveConnection	IDBCreateSession IDBInitialize	Sí
	BOF	(Específico de ADO)	
	Bookmark	IAccessor IRowsetLocate	Sí (Servicio de cursor)
	CacheSize	cRows in IRowsetLocate IRowset	Sí
	CursorType adOpenDynamic adOpenForwardOnly adOpenKeySet adOpenStatic	ICommandProperties	 No Sí No Sí (Servicio de cursor)
	EditMode	IRowsetUpdate	Sí (Servicio de cursor)
	EOF	(Específico de ADO)	
	Filter	IRowsetLocate IRowsetView IRowsetUpdate IViewChapter IViewFilter	No
	LockType	ICommandProperties	No
	MarshallOption		No
	MaxRecords	ICommandProperties IOpenRowset	Sí
	PageCount	IRowsetScroll	Sí (Servicio de cursor)
	PageSize	(Específico de ADO)	
	Sort	(Específico de ADO)	
	Source	(Específico de ADO)	
	State	(Específico de ADO)	
	Status	IRowsetUpdate	Sí (Servicio de cursor)

Tabla 28. Métodos y propiedades ADO soportados por IBM OLE DB Provider para DB2 (continuación)

ADO	Método/Propiedad	Interfaz/Propiedad OLE DB	Soporte de IBM OLE DB
Grupo de RecordSet	Fields	IColumnInfo	Sí
	Properties	IDBProperties IRowsetInfo::GetProperties	Sí

Aplicaciones C y C++

Las secciones siguientes describen consideraciones sobre aplicaciones C y C++.

Compilación y enlace de aplicaciones C/C++ e IBM OLE DB Provider

Las aplicaciones C/C++ que utilizan la constante CLSID_IBMDADB2 deben incluir el archivo `ibmdadb2.h`, que se encuentra en el directorio `SQLLIB\include`. Estas aplicaciones deben definir DBINITCONSTANTS antes de la sentencia include. El siguiente ejemplo muestra la secuencia correcta de sentencias:

```
#define DBINITCONSTANTS
#include "ibmdadb2.h"
```

Conexiones con fuentes de datos en aplicaciones C/C++ mediante IBM OLE DB Provider

Para conectar con una fuente de datos DB2 mediante IBM OLE DB Provider para DB2 en una aplicación C/C++, utilice una de las dos interfaces principales de OLE DB, `IDBPromptInitialize` o `IDataInitialize`. El hecho de conectar con la fuente de datos de este modo garantiza el acceso de la aplicación a cursores desplazables actualizables, en lugar de a los cursores de solo lectura y de solo avance disponibles de forma nativa. Si IBMDADB2 se crea directamente llamando a la API COM CoCreateInstance, los cursores disponibles serán de solo lectura y de solo avance. El componente OLE DB Service expone la interfaz `IDataInitialize` y el componente Data Links expone la interfaz `IDBPromptInitialize`.

Conceptos relacionados:

- "Conexiones a fuentes de datos mediante IBM OLE DB Provider" en la página 409

Tareas relacionadas:

- "Creación de aplicaciones ADO con Visual C++" en el manual *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

Cursores desplazables actualizables en aplicaciones ATL e IBM OLE DB Provider

Si una aplicación ATL (Active Template Library) necesita cursores desplazables actualizables y su Objeto Consumidor de acceso a datos se ha generado mediante ATL COM AppWizard, la aplicación debe utilizar el método `OpenWithServiceComponents` en lugar de la llamada al método `Open` (que es la que el asistente genera por omisión). Si se utiliza el método `Open`, los cursores disponibles serán de solo lectura y de solo avance. El siguiente ejemplo muestra cómo utilizar el método `OpenWithServiceComponents`:

```
// La siguiente línea la genera el asistente en el método OpenDataSource
//  hr = db.Open(_T("IBMDADB2"), &dbinit);
// Sustitúyala por lo siguiente:
hr = db.OpenWithServiceComponents(_T("IBMDADB2"), &dbinit);
```

Transacciones distribuidas MTS y COM+

Las secciones siguientes describen consideraciones sobre las transacciones distribuidas MTS y COM+.

Soporte de transacciones distribuidas MTS y COM+ e IBM OLE DB Provider

Las aplicaciones OLE DB que se ejecutan en un entorno Microsoft Transaction Server (MTS) en Windows NT o en un entorno Component Services (COM+) en Windows 2000 pueden utilizar la interfaz `ITransactionJoin` para participar en transacciones distribuidas con varios servidores de bases de datos DB2 Universal Database, de sistema principal y iSeries, así como otros gestores de recursos que cumplan con las especificaciones MTS/COM+.

Requisitos previos:

Para poder utilizar el soporte de transacciones distribuidas MTS o COM+ que ofrece IBM OLE DB Provider para DB2, asegúrese de que el servidor cumple los siguientes requisitos previos.

Nota: estos requisitos sólo afectan a la máquina Windows en la que está instalado el cliente DB2.

- Windows NT con MTS al nivel de Versión 2.0 con Microsoft Hotfix 0772 o posterior, o Windows 2000

  MTS Versión 2.0 para Windows NT está disponible como parte de Windows NT 4.0 Option Pack. Puede bajar el Option Pack de:

  `http://www.microsoft.com/ntserver/nts/downloads/recommended/NT4OptPk/`

Habilitación del soporte de MTS en DB2 Universal Database para aplicaciones C/C++

Para ejecutar una aplicación C o C++ en modalidad transaccional MTS o COM+, puede crear la instancia de fuente de datos IBMDABD2 mediante la interfaz `DataLink`. También podría utilizar `CoCreateInstance`, obtener un objeto de sesión y utilizar `JoinTransaction`. Consulte la descripción sobre cómo conectar una aplicación C o C++ a una fuente de datos para obtener más información.

Para ejecutar una aplicación ADO en modalidad transaccional MTS o COM+, consulte la descripción sobre cómo conectar una aplicación C o C++ a una fuente de datos.

Para utilizar un componente de un paquete MTS o COM+ en modalidad transaccional, establezca para la propiedad Transactions del componente uno de los siguientes valores:

- "Required"
- "Required New"
- "Supported"

Para obtener información sobre estos valores, consulte la documentación de MTS.

Parte 5. Conceptos generales sobre aplicaciones DB2

Capítulo 15. Soporte de idiomas nacionales

Visión general del orden de clasificación

Las secciones siguientes describen órdenes de clasificación y cómo se realizan las comparaciones de caracteres.

Órdenes de clasificación

El gestor de bases de datos compara datos de caracteres utilizando un *orden de clasificación*. Es una clasificación correspondiente a conjuntos de caracteres que determina si un determinado carácter se clasifica por encima, por debajo o igual que otro.

Nota: los datos de series de caracteres definidos con el atributo FOR BIT DATA y los datos BLOB se clasifican utilizando el orden de clasificación binaria.

Por ejemplo, se puede utilizar un orden de clasificación para indicar que las versiones en minúsculas y en mayúsculas de un determinado carácter se tienen que clasificar de igual forma.

El gestor de bases de datos permite que se creen bases de datos con órdenes de clasificación personalizados. Las secciones siguientes le ayudan a determinar y a implantar un determinado orden de clasificación para una base de datos.

Cada carácter de un solo byte de una base de datos se representa de forma interna como un número exclusivo comprendido entre 0 y 255 (en notación hexadecimal, entre X'00' y X'FF'). Se hace referencia a este número como *punto de código* del carácter; la asignación de números a caracteres en un conjunto se denomina de forma colectiva *página de códigos*. Un orden de clasificación es una correlación entre el punto de código y la posición deseada de cada carácter en un orden clasificado. El valor numérico de la posición se denomina *peso* del carácter en el orden de clasificación. En el orden de clasificación más sencillo, los pesos son idénticos a los puntos de código. Esto se denomina el *orden de identidad*.

Por ejemplo, supongamos que los caracteres B y b tienen los puntos de código X'42' y X'62' respectivamente. Si (según la tabla de orden de clasificación), ambos tienen un peso de clasificación de X'42' (B), se clasifican igual. Si el peso de clasificación correspondiente a B es X'9E' y el correspondiente a b es X'9D', b se clasificará antes que B. La tabla de orden de clasificación especifica el peso de cada carácter. La tabla es distinta de una página de códigos, la cual especifica el punto de código de cada carácter.

Supongamos que tenemos el siguiente ejemplo. Los caracteres ASCII comprendidos entre A y Z se representan mediante los valores comprendidos entre X'41' y X'5A'. Para describir un orden de clasificación en el que estos caracteres se clasifican de forma consecutiva (sin caracteres intermedios), puede escribir X'41', X'42', ... X'59', X'5A'.

El valor hexadecimal de un carácter de varios bytes también se utiliza como el peso. Por ejemplo, supongamos que los puntos de código correspondientes a los caracteres de doble byte A y B son X'8260' y X'8261' respectivamente; se utilizan los pesos de clasificación correspondientes a X'82', X'60' y X'61' para clasificar estos dos caracteres se acuerdo con sus puntos de código.

No hace falta que los pesos de un orden de clasificación sean exclusivos. Por ejemplo, podría asignar a letras mayúsculas y a sus equivalentes minúsculas el mismo peso.

La especificación de un orden de clasificación se puede simplificar si el orden de clasificación proporciona pesos para los 256 puntos de código. El peso de cada carácter se puede determinar mediante el punto de código del carácter.

En todos los casos, DB2 utiliza la tabla de clasificación especificada en el momento de creación de la base de datos. Si desea que los caracteres de varios bytes se clasifiquen del modo en que aparecen en su tabla de puntos de código, debe especificar IDENTITY como el orden de clasificación al crear la base de datos.

Nota: para caracteres de doble byte y Unicode en campos GRAPHIC, el orden de clasificación siempre es IDENTITY.

Una vez definido un orden de clasificación, todas las futuras comparaciones de caracteres correspondientes a dicha base de datos se realizará con este orden de clasificación. Excepto para los datos definidos como FOR BIT DATA o datos BLOB, el orden de clasificación se utilizará para todas las comparaciones de SQL y cláusulas ORDER BY y también al configurar índices y estadísticas.

Se pueden producir problemas potenciales en los siguientes casos:

- Una aplicación fusiona datos clasificados de una base de datos con datos de aplicación que se clasificaron utilizando otro orden de clasificación.
- Una aplicación fusiona datos clasificados de una base de datos con datos clasificados de otra, pero las bases de datos tienen distintos órdenes de clasificación.
- Una aplicación realiza supuestos sobre datos clasificados que no resultan ciertos para el orden de clasificación relevante. Por ejemplo, que los números se clasifican por debajo que los caracteres alfabéticos puede resultar o no cierto para un determinado orden de clasificación.

Un punto final a tener en cuenta es que los resultados de cualquier clasificación basada en una comparación directa de puntos de código de caracteres sólo coincidirá con los resultados de consultas que estén clasificados utilizando un orden de clasificación de identidad.

Conceptos relacionados:

- "conversión de caracteres" en el manual *Consulta de SQL, Volumen 1*
- "Comparaciones de caracteres basadas en órdenes de clasificación" en la página 424

Comparaciones de caracteres basadas en órdenes de clasificación

Una vez establecido un orden de clasificación, la comparación de caracteres se lleva a cabo comparando los pesos de dos caracteres, en lugar de comparando directamente sus valores de punto de código.

Si se utilizan pesos que no son exclusivos, es posible que caracteres que no son idénticos se comparen y queden iguales. Por este motivo, la comparación de series se puede convertir en un proceso de dos fases:

1. Comparar los caracteres de cada serie según sus pesos.
2. Si el paso 2 da como resultado igualdad, comparar los caracteres de cada serie según sus valores de punto de código.

Si el orden de clasificación contiene 256 pesos exclusivos, sólo hay que llevar a cabo el primer paso. Si el orden de clasificación es el orden de identidad, sólo hay que llevar a cabo el segundo paso. En cualquier caso, hay una mejora del rendimiento.

Conceptos relacionados:

- "conversión de caracteres" en el manual *Consulta de SQL, Volumen 1*

Comparaciones que no dependen de mayúsculas y minúsculas mediante la función TRANSLATE

Para realizar comparaciones de caracteres que no dependen de mayúsculas y minúsculas, puede utilizar la función TRANSLATE para seleccionar y comparar datos de columnas de mayúsculas y minúsculas combinadas, convirtiéndolos todos a mayúsculas (sólo para la comparación). Supongamos que tenemos los datos siguientes:

```
Abel
abels
ABEL
abel
ab
Ab
```

La siguiente sentencia SELECT:

```
SELECT c1 FROM T1 WHERE TRANSLATE(c1) LIKE 'AB%'
```

devuelve

```
ab
Ab
abel
Abel
ABEL
abels
```

También podría especificar la siguiente sentencia SELECT al crear la vista ″v1″, realizar todas las comparaciones contra la vista en mayúsculas y solicitar operaciones INSERT de la tabla en mayúsculas y minúsculas combinadas:

```
CREATE VIEW v1 AS SELECT TRANSLATE(c1) FROM T1
```

A nivel de base de datos, puede definir el orden de clasificación como parte de la API sqlecrea. - Crear base de datos. Esto le permite decidir si ″a″ se tiene que procesar antes que ″A″, si ″A″ se tiene que procesar después que ″a″ o si se tienen que procesar con el mismo peso. Esto las convertiría en iguales si se realiza la clasificación mediante la cláusula ORDER BY. ″A″ siempre irá antes que ″a″, porque son equivalentes en cualquier sentido. La única base por la que clasificar es el valor hexadecimal.

Por lo tanto

```
SELECT c1 FROM T1 WHERE c1 LIKE 'ab%'
```

devuelve

```
ab
abel
abels
```

y

```
SELECT c1 FROM T1 WHERE c1 LIKE 'A%'
```

devuelve

```
Abel
Ab
ABEL
```

La sentencia siguiente

```
SELECT c1 FROM T1 ORDER BY c1
```

devuelve

```
ab
Ab
abel
Abel
ABEL
abels
```

Por lo tanto, quizás desee tener en cuenta la posibilidad de utilizar la función escalar TRANSLATE(), además de sqlecrea. Tenga en cuenta que sólo puede especificar un orden de clasificación mediante sqlecrea. No puede especificar un orden de clasificación desde el procesador de línea de mandatos (CLP).

También puede utilizar la función UCASE del siguiente modo, pero tenga en cuenta que DB2 realiza una exploración de tabla en lugar de utilizar un índice para esta operación select:

```
SELECT * FROM EMP WHERE UCASE(JOB) = 'NURSE'
```

Consulta relacionada:

- "Función escalar TRANSLATE" en el manual *Consulta de SQL, Volumen 1*
- "Función escalar UCASE o UPPER" en el manual *Consulta de SQL, Volumen 1*
- "sqlecrea - Create Database" en el manual *Administrative API Reference*

Diferencias entre los órdenes de clasificación de EBCDIC y ASCII

El orden en que se clasifican los datos de una base de datos depende del orden de clasificación definido para la base de datos. Por ejemplo, supongamos que la base de datos A utiliza el orden de clasificación por omisión de la página de códigos EBCDIC y que la base de datos B utiliza el orden de clasificación por omisión de la página de códigos ASCII. Los órdenes de clasificación de estas dos bases de datos diferirían, tal como se muestra en el siguiente ejemplo:

```
SELECT.....
  ORDER BY COL2

Orden clasif. EBCDIC     Orden clasif. ASCII

COL2                     COL2
----                     ----
V1G                      7AB
Y2W                      V1G
7AB                      Y2W
```

Figura 7. Ejemplo de cómo difiere un orden de clasificación basado en EBCDIC de un orden de clasificación basado en ASCII

De forma similar, las comparaciones de caracteres de una base de datos dependen del orden de clasificación definido para dicha base de datos. De este modo, si la base de datos A utiliza el orden de clasificación por omisión de la página de códigos EBCDIC y la base de datos B utiliza el orden de clasificación por omisión de la página de códigos ASCII, los resultados de

comparaciones de caracteres en las dos bases de datos diferirían. La diferencia es la siguiente:

```
SELECT.....
  WHERE COL2 > 'TT3'

Resultados EBCDIC        Resultados ASCII

COL2                     COL2
----                     ----
TW4                      TW4
X72                      X72
39G
```

Figura 8. Ejemplo de cómo una comparación de caracteres en un orden de clasificación basado en EBCDIC difiere de una comparación de caracteres en un orden de clasificación basado en ASCII

Si va a crear una base de datos federada, considere la posibilidad de especificar que el orden de clasificación coincida con el orden de clasificación de la fuente de datos. Este enfoque maximizará las oportunidades de "empuje" y posiblemente aumentará el rendimiento de las consultas.

Conceptos relacionados:

- "Guidelines for analyzing where a federated query is evaluated" en el manual *Administration Guide: Performance*

Orden de clasificación especificado cuando se crea la base de datos

El orden de clasificación correspondiente a una base de datos se especifica en el momento de creación de la base de datos. Una vez creada la base de datos, el orden de clasificación no se puede cambiar.

La API CREATE DATABASE acepta una estructura de datos denominada Bloque descriptor de base de datos (SQLEDBDESC). Puede definir su propio orden de clasificación dentro de esta estructura.

Nota: sólo puede definir su propio orden de clasificación para una base de datos de un solo byte.

Para especificar un orden de clasificación para una base de datos:

- Pase la estructura SQLEDBDESC que desee, o
- Pase un puntero NULL. Se utiliza el orden de clasificación del sistema operativo (basado en la página de códigos y la página de país/región actuales). Esto equivale a especificar para SQLDBCSS el valor SQL_CS_SYSTEM (0).

La estructura SQLEDBDESC contiene:

SQLDBCSS Un entero de 4 bytes que indica la fuente del orden de clasificación de la base de datos. Los valores válidos son:

SQL_CS_SYSTEM
Se utiliza el orden de clasificación del sistema operativo (basado en la página de códigos y la página de país/región actuales).

SQL_CS_SYSTEM_NLSCHAR
Orden de clasificación del usuario que utiliza la versión NLS de rutinas de comparación para tipos de caracteres

SQL_CS_IDENTITY_16BIT
Una base de datos Unicode se puede crear con la opción de clasificación SQL_CS_IDENTITY_16BIT. SQL_CS_IDENTITY_16BIT difiere de la opción de clasificación SQL_CS_NONE por omisión en que los datos CHAR, VARCHAR, LONG VARCHAR y CLOB de la base de datos Unicode se clasificarán utilizando el orden binario CESU-8 en lugar del orden binario UTF-8. CESU-8 es un Esquema de codificación de compatibilidad para UTF-16: 8 bits y, en lo que a este documento se refiere, su especificación está contenida en el Borrador de informe técnico de Unicode número 26 disponible en el sitio web de Unidoce Technical Consortium (www.unicode.org). CESU-8 es idéntico en binario a UTF-8, excepto los caracteres complementarios de Unicode, es decir, los caracteres que están definidos fuera de Basic Multilingual Plane de 16 bits (BMP o Plane 0). En codificación UTF-8, un carácter complementario se representa mediante una secuencia de 4 bytes, pero el mismo carácter en CESU-8 necesita dos secuencias de 3 bytes. En una base de datos Unicode, los datos CHAR, VARCHAR, LONG VARCHAR y CLOB se almacenan en UTF-8 y los datos GRAPHIC, VARGRAPHIC, LONG VARGRAPHIC y DBCLOB se almacenan en UCS-2. Para la clasificación SQL_CS_NONE, los caracteres no complementarios en UTF-8 y UCS-2 tienen la misma clasificación binaria, pero los caracteres

complementarios en UTF-8 se clasifican de forma distinta a los mismos caracteres en UCS-2. SQL_CS_IDENTITY_16BIT asegura que todos los caracteres, complementarios y no complementarios, de una base de datos Unicode DB2 tienen la misma clasificación binaria.

SQL_CS_USER
El orden de clasificación se especifica mediante el valor del campo SQLDBUDC.

SQL_CS_NONE
El orden de clasificación es la clasificación de identidad. Las series se comparan byte a byte, empezando por el primer byte, utilizando una sencilla comparación de puntos de código.

Nota: estas constantes se definen en el archivo include SQLENV.

SQLDBUDC Un campo de 256 bytes. El byte número *n* contiene el peso de clasificación del carácter número *n* de la página de códigos de la base de datos. Si SQLDBCSS no es igual a SQL_CS_USER, este campo se pasa por alto.

Consulta relacionada:

- "sqlecrea - Create Database" en el manual *Administrative API Reference*

Órdenes de clasificación de ejemplo

Se proporcionan varios órdenes de clasificación de ejemplo (como archivos include) para facilitar la creación de bases de datos utilizando los órdenes de clasificación EBCDIC en lugar del orden de clasificación por omisión de la estación de trabajo.

Los órdenes de clasificación de estos archivos include se pueden especificar en el campo SQLDBUDC de la estructura SQLEDBDESC. También se pueden utilizar como modelos para la construcción de otros órdenes de clasificación.

Hay archivos include que contienen órdenes de clasificación disponibles para los siguientes lenguajes principales:

- C/C++
- COBOL
- FORTRAN

Consulta relacionada:

- "Archivos include para C y C++" en la página 177
- "Archivos include para COBOL" en la página 234
- "Archivos include para FORTRAN" en la página 263

Páginas de códigos y entornos locales

Las secciones siguientes describen páginas de códigos y cómo se obtienen páginas de códigos y entornos locales.

Derivación de valores de página de códigos

La *página de códigos de la aplicación* se deriva del entorno activo cuando se realiza la conexión con la base de datos. Si la variable de registro DB2CODEPAGE está definida, se toma este valor como página de códigos de la aplicación. Sin embargo, no es necesario definir la variable de registro DB2CODEPAGE porque DB2 determinará el valor adecuado de página de códigos a partir del sistema operativo. Si se define para la variable de registro DB2CODEPAGE un valor incorrecto se pueden producir resultados inesperados.

La *página de códigos de la base de datos* se deriva del valor especificado (de forma explícita o por omisión) en el momento en que se crea la base de datos. Por ejemplo, lo siguiente define cómo se determina el *entorno activo* en distintos sistemas operativos:

UNIX En sistemas operativos basados en UNIX, el entorno activo se determina a partir del valor de entorno local (locale) que incluye información sobre idioma, territorio y juego de códigos.

Sistemas operativos Windows
Para todos los sistemas operativos Windows, si la variable de entorno DB2CODEPAGE no está definida, la página de códigos se deriva del valor de página de códigos ANSI del Registro.

Consulta relacionada:

- "Supported territory codes and code pages" en el manual *Administration Guide: Planning*

Derivación de entornos locales en programas de aplicación

Los entornos locales se implantan de una manera en sistemas Windows y de otra en sistemas basados en UNIX. Hay dos entornos locales en sistemas basados en UNIX:

- El entorno local del entorno le permite especificar el idioma, el símbolo de moneda, etc. que desea utilizar.
- El entorno local del programa contiene el idioma, símbolo de moneda, etc. actuales de un programa que se está ejecutando.

En sistemas Windows, las preferencias culturales se pueden definir mediante Configuración regional en el Panel de control. Sin embargo, no hay ningún entorno local del entorno como el de sistemas basados en UNIX.

Cuando se inicia el programa, obtiene un entorno local C por omisión. *No* obtiene una copia del entorno local del entorno. Si define como entorno local del programa cualquier entorno local que no sea ″C″, DB2 Universal Database utiliza el entorno local del programa actual para determinar los valores de página de códigos y de territorio correspondientes al entorno de la aplicación. Si no es así, estos valores se obtienen del entorno del sistema operativo. Tenga en cuenta que `setlocale()` no está libre de hebras y que si emite `setlocale()` desde dentro de la aplicación, el nuevo entorno local se establece para el proceso entero.

Cómo DB2 deriva entornos locales

En sistemas basados en UNIX, el entorno local activo que utiliza DB2 se determina a partir de la parte LC_CTYPE del valor del entorno local. Para ver detalles, consulte la documentación de NLS correspondiente a su sistema operativo.

- Si LC_CTYPE del entorno local del programa tiene un valor distinto de C, DB2 utilizará este valor para determinar la página de códigos de la aplicación, correlacionándolo con su página de códigos correspondiente.
- Si LC_CTYPE tiene el valor C (el entorno local C), DB2 definirá el entorno local del programa de acuerdo con el entorno local del entorno, mediante la función `setlocale()`.
- Si LC_CTYPE sigue teniendo el valor C, DB2 adoptará el valor por omisión de entorno Inglés EE.UU. y la página de códigos 819 (ISO 8859-1).
- Si LC_CTYPE ya no tiene el valor C, se utilizará su nuevo valor para correlacionarlo con una página de códigos correspondiente.

Consulta relacionada:

- “Supported territory codes and code pages” en el manual *Administration Guide: Planning*

Consideraciones sobre aplicaciones

Las secciones siguientes describen consideraciones que debe tener en cuenta al codificar una aplicación.

Consideraciones sobre el soporte de idiomas nacionales y sobre el desarrollo de aplicaciones

Las series de caracteres constantes en sentencias de SQL estático se convierten en el momento de la vinculación de la página de códigos de la aplicación a la página de códigos de la base de datos, y luego se utilizan en el momento de la ejecución en esta representación de página de códigos de la base de datos. Para evitar estas conversiones si no se desean, puede utilizar variables del lenguaje principal en lugar de constantes de series.

Si el programa contiene series de caracteres constantes, debe precompilar, vincular, compilar y ejecutar la aplicación utilizando la misma página de códigos. Para una base de datos Unicode, debe utilizar variables del lenguaje principal en lugar de utilizar constantes de series. La razón de esta recomendación es que las conversiones de datos que realiza el servidor se pueden producir en la fase de vinculación y en la de ejecución. Esto puede resultar un problema si se utilizan series de caracteres contantes dentro del programa. Estas series incorporadas se convierten en el momento de la vinculación según la página de códigos que está en vigor durante la fase de vinculación. Los caracteres ASCII de siete bits son comunes a todas las páginas de códigos soportadas por DB2 Universal Database y no ocasionan problemas. Para caracteres no ASCII, los usuarios se deben asegurar de que se utilizan las mismas tablas de conversión durante la vinculación y la ejecución con la misma página de códigos activa.

Se supone que cualquier dato externo que obtiene la aplicación está en la página de códigos de la aplicación. Esto incluye datos obtenidos de un archivo o de entrada del usuario. Asegúrese de que los datos procedentes de fuentes externas a la aplicación utilizan la misma página de códigos que la aplicación.

Si utiliza variables del lenguaje principal que utilizan datos gráficos en aplicaciones C o C++, hay temas especiales relacionados con el precompilador, el rendimiento de la aplicación y el diseño de la aplicación que debe tener en cuenta. Si utiliza juegos de códigos EUC en las aplicaciones, consulte los temas relacionados para ver las directrices generales.

Cuando desarrolle una aplicación, debe revisar los temas que siguen a este. Si no sigue las recomendaciones que se describen en estos temas, se pueden producir condiciones inesperadas. Estas condiciones las puede detectar el gestor de bases de datos, así que emitirá un mensaje de error o de aviso. Por ejemplo, una aplicación C contiene las siguientes sentencias de SQL que operan contra una tabla T1 con una columna definida como C1 CHAR(20):

```
(0)  EXEC SQL CONNECT TO GLOBALDB;
(1)  EXEC SQL INSERT INTO T1 VALUES ('a-constant');
     strcpy(sqlstmt, "SELECT C1 FROM T1 WHERE C1='a-constant');
```

```
     (2)  EXEC SQL PREPARE S1 FROM :sqlstmt;
Donde:
    página de códigos de la aplicación en el momento de la vinculación = x
    página de códigos de la aplicación en el momento de la ejecución = y
    página de códigos de la base de datos = z
```

En el momento de la vinculación '*a-constant*' en la sentencia (1) se convierte de la página de códigos **x** a la página de códigos **z**. Esta conversión se representa como (x→z).

En el momento de la ejecución, '*a-constant*' (x→z) se inserta en la tabla cuando se ejecuta la sentencia (1). Sin embargo, la cláusula WHERE de la sentencia (2) se ejecutará con '*a-constant*' (y→z). Si los puntos de código de la constante son tales que las dos conversiones (x→z e y→z) generan resultados diferentes, la operación SELECT de la sentencia (2) no podrá recuperar los datos insertados por la sentencia (1).

Conceptos relacionados:

- “Variables gráficas del lenguaje principal en C y C++” en la página 191
- “Derivación de valores de página de códigos” en la página 430
- “Consideraciones sobre los juegos de códigos EUC y UCS-2 de japonés y chino tradicional” en la página 444

Soporte de idiomas nacionales y sentencias de SQL

La codificación de sentencias de SQL no depende del idioma. Las palabras clave de SQL se deben escribir tal como se muestra, aunque se pueden escribir en mayúsculas, en minúsculas o en una combinación de ambas. Los caracteres de los nombres de objetos de base de datos, variables del lenguaje principal y etiquetas de programa que aparecen en una sentencia de SQL deben recibir soporte de la página de códigos de la aplicación.

El servidor no convierte nombres de archivo. Para codificar un nombre de archivo, puede utilizar el juego invariante ASCII o proporcionar la vía de acceso en los valores hexadecimales que están físicamente almacenados en el sistema de archivos.

En un entorno de varios bytes, hay cuatro caracteres que se consideran especiales que no pertenecen al juego de caracteres invariante. Estos caracteres son:

- Los caracteres de porcentaje de doble byte y de subrayado de doble byte que se utilizan en proceso LIKE.
- El carácter de espacio de doble byte que se utiliza, entre otras cosas, para rellenar con blancos series gráficas.

- El carácter de sustitución de doble byte, que se utiliza como sustitución durante la conversión de página de códigos cuando no existe ninguna correlación entre una página de códigos fuente y una página de códigos de destino.

Los puntos de código para cada uno de estos caracteres, por página de códigos, son los siguientes:

Tabla 29. Puntos de código para caracteres especiales de doble byte

Página de códigos	Porcentaje de doble byte	Subrayado de doble byte	Espacio de doble byte	Carácter de sustitución de doble byte
932	X'8193'	X'8151'	X'8140'	X'FCFC'
938	X'8193'	X'8151'	X'8140'	X'FCFC'
942	X'8193'	X'8151'	X'8140'	X'FCFC'
943	X'8193'	X'8151'	X'8140'	X'FCFC'
948	X'8193'	X'8151'	X'8140'	X'FCFC'
949	X'A3A5'	X'A3DF'	X'A1A1'	X'AFFE'
950	X'A248'	X'A1C4'	X'A140'	X'C8FE'
954	X'A1F3'	X'A1B2'	X'A1A1'	X'F4FE'
964	X'A2E8'	X'A2A5'	X'A1A1'	X'FDFE'
970	X'A3A5'	X'A3DF'	X'A1A1'	X'AFFE'
1381	X'A3A5'	X'A3DF'	X'A1A1'	X'FEFE'
1383	X'A3A5'	X'A3DF'	X'A1A1'	X'A1A1'
13488	X'FF05'	X'FF3F'	X'3000'	X'FFFD'
1363	X'A3A5'	X'A3DF'	X'A1A1'	X'A1E0'
1386	X'A3A5'	X'A3DF'	X'A1A1'	X'FEFE'
5039	X'8193'	X'8151'	X'8140'	X'FCFC'

Para bases de datos Unicode, el espacio GRAPHIC es X'0020', que es distinto del espacio GRAPHIC X'3000' que se utiliza para bases de datos euc-Japan y euc-Taiwan. Tanto X'0020' como X'3000' son caracteres de espacio en estándar Unicode. La diferencia en los puntos de código de espacio GRAPHIC se debe tener en cuenta cuando se comparan datos procedentes de estas bases de datos EUC con datos procedentes de una base de datos Unicode.

Consulta relacionada:

- “Predicado LIKE” en el manual *Consulta de SQL, Volumen 1*
- “Juegos de caracteres de Extended UNIX Code (EUC)” en la página 442

Procedimientos almacenados y UDF remotos

Cuando se codifican procedimientos almacenados que se van a ejecutar de forma remota, se deben tener en cuenta las siguientes consideraciones:

- Los datos de un procedimiento almacenado deben estar en la página de códigos de la base de datos.
- Los datos que se pasan a un procedimiento almacenado o que se reciben de este mediante una SQLDA con un tipo de datos de carácter deben contener realmente datos de carácter. Los datos numéricos y las estructuras de datos nunca se deben pasar con un tipo de carácter si la página de códigos de la aplicación cliente es distinta de la página de códigos de la base de datos. El motivo es que el servidor convertirá todos los datos de carácter de una SQLDA. Para evitar la conversión de caracteres, puede pasar datos definiéndolos en formato de serie binaria utilizando un tipo de datos BLOB o definiendo los datos de carácter como FOR BIT DATA.

Por omisión, cuando se invocan UDF y procedimientos almacenados DARI de DB2, estos se ejecutan bajo un entorno de idioma nacional por omisión, que puede no coincidir con el entorno de idioma nacional de la base de datos. Por lo tanto, la utilización de operaciones específicas de país/región o de página de códigos, como funciones y variables del lenguaje principal gráficas `wchar_t` de C, puede no funcionar según lo esperado. Se tiene que asegurar de que, si es necesario, se inicialice el entorno correcto cuando invoque el procedimiento almacenado o UDF.

Consideraciones sobre nombres de paquetes en entornos de páginas de códigos combinadas

Los nombres de paquetes se determinan cuando se invoca la API o mandato PRECOMPILE PROGRAM. Por omisión, se generan tomando los ocho primeros bytes del archivo fuente del programa de aplicación (sin la extensión de archivos) y se convierten a mayúsculas. También se puede definir un nombre de forma explícita. Independientemente del origen de un nombre de paquete, si trabaja en un entorno de páginas de códigos que no son iguales, los caracteres de los nombres de paquetes deben estar en un juego de caracteres invariante. De lo contrario pueden producirse problemas relacionados con la modificación del nombre de paquete. El gestor de bases de datos no podrá encontrar el paquete correspondiente a la aplicación o una herramienta del cliente no mostrará el nombre correcto del paquete.

Se producirá una modificación de nombre de paquete debida a conversión de caracteres si alguno de los caracteres del nombre del paquete no se correlaciona directamente con un carácter válido de la página de códigos de la base de datos. En estos casos, un carácter de sustitución sustituye al carácter que no se ha convertido. Tras dicha modificación, es posible que el nombre del paquete, cuando se vuelve a convertir a la página de códigos de la

aplicación, no coincida con el nombre original del paquete. Un ejemplo de un caso en que no se desea este comportamiento es cuando utiliza el Centro de control para listar paquetes y para trabajar con los mismos. Es posible que los nombres de paquetes que se muestran no coincidan con los nombres esperados.

Para evitar problemas de conversión con nombres de paquetes, asegúrese de que sólo se utilizan caracteres que son válidos para la página de códigos de la aplicación y para la de la base de datos.

Página de códigos activa para precompilación y vinculación

En el momento de la precompilación / vinculación, el precompilador es la aplicación que se ejecuta. Se utiliza la página de códigos activa cuando se realizó la conexión con la base de datos antes de la petición de precompilación para sentencias precompiladas y para los datos que se devuelven en la SQLCA.

Conceptos relacionados:

- "Página de códigos activa para la ejecución de aplicaciones" en la página 436

Página de códigos activa para la ejecución de aplicaciones

En el momento de la ejecución, la página de códigos activa de la aplicación de usuario cuando se realiza la conexión con la base de datos permanece en vigor mientras dura la conexión. Todos los datos se interpretan según esta página de códigos; esto incluye sentencias de SQL dinámico, datos de entrada del usuario, datos de salida del usuario y campos de caracteres de la SQLCA.

Conceptos relacionados:

- "Página de códigos activa para precompilación y vinculación" en la página 436

Conversión de caracteres entre distintas páginas de códigos

Lo ideal para conseguir un rendimiento óptimo sería que las aplicaciones utilizaran siempre la misma página de códigos que la base de datos. Sin embargo, esto no siempre es posible o resulta práctico. Los productos DB2 proporcionan soporte para la conversión de página de códigos que permite que la aplicación y la base de datos utilicen distintas páginas de códigos. Los caracteres de una página de códigos se deben correlacionar con la otra página de códigos para mantener la integridad de los datos.

Cuándo se produce la conversión de página de códigos

La conversión de página de códigos se puede producir en las siguientes situaciones:

- Cuando un cliente o aplicación que accede a una base de datos se ejecuta en una página de códigos distinta de la de la base de datos:

  Esta conversión se producirá en el cliente de aplicación para ambas conversiones, de la página de códigos de la aplicación a la de la base de datos y de la página de códigos de la base de datos a la de la aplicación.

  En algunas situaciones puede minimizar o eliminar la conversión de caracteres cliente/servidor. Por ejemplo, puede:

  – Crear una base de datos en Windows NT que utilice la página de códigos 850 para que coincida con un entorno de aplicaciones cliente de Windows que utiliza predominantemente la página de códigos 850.

    Si se utiliza una aplicación ODBC de Windows con el controlador ODBC de DB2 de IBM en el cliente de bases de datos de Windows, este problema se puede mitigar utilizando las palabras clave TRANSLATEDLL y TRANSLATEOPTION en el archivo `odbc.ini` o `db2cli.ini`.

  – Crear una base de datos en AIX que utilice la página de códigos 850 para que coincida con un entorno de aplicaciones cliente que utiliza predominantemente la página de códigos 850.

- Cuando un cliente o aplicación que importa un archivo PC/IXF se ejecuta en una página de códigos distinta de la del archivo que se va a importar.

  Esta conversión de datos se producirá en la máquina cliente de bases de datos antes de que el cliente acceda al servidor de bases de datos. Es posible que se lleve a cabo una conversión de datos adicional si la aplicación se ejecuta en una página de códigos distinta de la de la base de datos (tal como se ha indicado en el punto anterior).

  La conversión de datos, si se realiza, también depende del modo en que se haya llamado al programa de utilidad de importación.

- Cuando se utiliza DB2 Connect para acceder a datos en un sistema principal, AS/400 o servidor iSeries. En este caso, el receptor de los datos convierte los datos de caracteres. Por ejemplo, DB2 para MVS/ESA convierte al identificador de juego de caracteres codificados (CCSID) adecuado de MVS los datos que se envían a DB2 para MVS/ESA. DB2 Connect convierte los datos que se envían a la máquina DB2 Connect desde DB2 para MVS/ESA.

No se producirá conversión de caracteres para:

- Nombres de archivos. Debe utilizar el juego invariable ASCII para nombres de archivos o debe proporcionar el nombre de archivo en los valores hexadecimales que están físicamente almacenados en el sistema de archivos.

Tenga en cuenta que, si incluye un nombre de archivo como parte de una sentencia de SQL, se convierte como parte de la conversión de la sentencia.

- Los datos destinados a una columna que tiene asignado el atributo FOR BIT DATA o que proceden de ésta o los datos que se utilizan en una operación de SQL cuyo resultado es datos FOR BIT o BLOB. En estos casos, los datos se tratan como una corriente de bytes y no se produce ninguna conversión.

  Nota: un literal insertado en una columna definida como FOR BIT DATA se podría convertir si dicho literal formara parte de una sentencia de SQL que se ha convertido.
- Un producto o plataforma DB2 que no dé soporte, o que no tenga el soporte instalado, a la combinación deseada de páginas de códigos. En este caso, se devuelve un SQLCODE -332 (SQLSTATE 57017) si intenta ejecutar la aplicación.

Conceptos relacionados:

- "conversión de caracteres" en el manual *Consulta de SQL, Volumen 1*

Sustitución de caracteres durante conversiones de páginas de códigos

Si la aplicación realiza una conversión de una página de códigos a otra, es posible que uno o más caracteres no estén representados en la página de códigos de destino. Si esto sucede, DB2 inserta un carácter de *sustitución* en la serie de destino en lugar del carácter que no tiene representación. El carácter de sustitución se considera una parte válida de la serie. En situaciones en las que se produce una sustitución, el indicador SQLWARN10 de la SQLCA adopta el valor 'W'.

Nota: cualquier otra conversión de caracteres resultante del uso de la opción del precompilador WCHARTYPE CONVERT no marcarán un aviso si se realiza alguna sustitución.

Conceptos relacionados:

- "Opción del precompilador WCHARTYPE en C y C++" en la página 211

Consulta relacionada:

- "Mandato PRECOMPILE" en el manual *Consulta de mandatos*

Conversiones de páginas de códigos soportadas

Cuando se produce una conversión de datos, la conversión tiene lugar de una *página de códigos fuente* a una *página de códigos de destino*.

La página de códigos fuente se determina a partir de la fuente de los datos; los datos procedentes de la aplicación tienen una página de códigos fuente

igual a la página de códigos de la aplicación y los datos procedentes de la base de datos tienen una página de códigos fuente igual a la página de códigos de la base de datos.

La determinación de la página de códigos de destino es más complicada; hay que tener en cuenta dónde se van a colocar los datos, incluidas las normas correspondientes a operaciones intermedias:

- Si los datos se mueven directamente desde una aplicación a una base de datos, sin que intervengan operaciones, la página de códigos de destino es la de la base de datos.
- Si los datos se importan en una base de datos desde un archivo PC/IXF, hay dos pasos de conversión de caracteres:
  1. Desde la página de códigos del archivo PC/IXF (página de códigos fuente) a la página de códigos de la aplicación (página de códigos de destino)
  2. Desde la página de códigos de la aplicación (página de códigos fuente) a la página de códigos de la base de datos (página de códigos de destino)

  Debe tener cuidado en situaciones en las que se pueden producir dos pasos de conversión. Para evitar una posible pérdida de datos de caracteres, asegúrese de seguir las conversiones de caracteres soportadas. Además, dentro de cada grupo, solo los caracteres que existen en ambas páginas de códigos, fuente y destino, tienen conversiones significativas. Se utilizan otros caracteres como *sustituciones* y sólo resultan útiles para convertir desde la página de códigos de destino de nuevo a la página de códigos fuente (y no necesariamente proporcionan conversiones significativas en el proceso de conversión de dos pasos mencionado anteriormente). Estos problemas se pueden evitar si la página de códigos de la aplicación coincide con la de la base de datos.
- Si los datos se derivan de operaciones realizadas en datos de caracteres, en los que la fuente puede ser la página de códigos de la aplicación, la de la base de datos, FOR BIT DATA o datos BLOB, la conversión de datos se basa en un conjunto de normas. Es posible que algunos de los elementos de datos, o todos ellos, se tengan que convertir a un resultado intermedio antes de que se pueda determinar la página de códigos de destino final.

Nota: las conversiones de páginas de códigos entre páginas de códigos de varios bytes, por ejemplo DBCS y EUC, pueden dar como resultado un aumento o reducción de la longitud de la serie.

Conceptos relacionados:

- "conversión de caracteres" en el manual *Consulta de SQL, Volumen 1*

- “Conversión de caracteres entre distintas páginas de códigos” en la página 436

Consulta relacionada:

- “Supported territory codes and code pages” en el manual *Administration Guide: Planning*

Factor de expansión de conversión de página de códigos

Si la aplicación completa satisfactoriamente un intento de establecer conexión con un servidor de base de datos DB2, debe tener en cuenta los siguientes campos en la SQLCA que se devuelve:

- El segundo símbolo del campo SQLERRMC (los símbolos se separan mediante X'FF') indica la página de códigos de la base de datos. El noveno símbolo del campo SQLERRMC indica la página de códigos de la aplicación. Al consultar la página de códigos de la aplicación y compararla con la página de códigos de la base de datos se indica a la aplicación si ha establecido una conexión que experimentará conversiones de caracteres.
- La primera y segunda entrada de la matriz SQLERRD. SQLERRD(1) contiene un valor entero igual al factor máximo esperado de expansión o contracción correspondiente a la longitud de datos de caracteres mixtos (tipos de datos CHAR) cuando se convierten a la página de códigos de la base de datos desde la página de códigos de la aplicación. SQLERRD(2) contiene un valor entero igual al factor máximo esperado de expansión o contracción correspondiente a la longitud de datos de caracteres mixtos (tipos de datos CHAR) cuando se convierten a la página de códigos de la aplicación desde la página de códigos de la base de datos. Un valor 0 ó 1 indica que no hay expansión; un valor mayor que 1 indica una posible expansión de la longitud; un valor negativo indica una posible contracción.

Las consideraciones correspondientes a datos de series gráficas no deberían tenerse en cuenta en situaciones de páginas de códigos distintas. Cada serie tiene siempre el mismo número de caracteres, independientemente de si los datos están en la página de códigos de la aplicación o en la de la base de datos.

Conceptos relacionados:

- “Desarrollo de aplicaciones en situaciones de páginas de códigos distintas” en la página 449

Consulta relacionada:

- “CONNECT (Tipo 1) sentencia” en el manual *Consulta de SQL, Volumen 2*
- “CONNECT (Tipo 2) sentencia” en el manual *Consulta de SQL, Volumen 2*

Juegos de caracteres DBCS

Cada página de códigos combinada de juego de caracteres de un solo byte (SBCS) o de juego de caracteres de doble byte (DBCS) permite puntos de código de caracteres de un solo byte y de doble byte. Esto se suele conseguir reservando un subconjunto de los 256 puntos de códigos disponibles de una tabla de códigos mixta para caracteres de un solo byte, con el resto de puntos de código no definidos o asignados al primer byte de puntos de código de doble byte. Estos puntos de código se muestran en la tabla siguiente.

Tabla 30. Puntos de código de juegos de caracteres mixtos

País/Región	Página de códigos mixta soportada	Puntos de código para caracteres de un solo byte	Puntos de código para el primer byte de caracteres de doble byte
Japón	932, 943	X'00'-X'7F', X'A1'-X'DF'	X'81'-X'9F', X'E0'-X'FC'
Japón	942	X'00'-X'80', X'A0'-X'DF', X'FD'-X'FF'	X'81'-X'9F', X'E0'-X'FC'
Taiwán	938 (*)	X'00'-X'7E'	X'81'-X'FC'
Taiwán	948 (*)	X'00'-X'80', X'FD', X'FE'	X'81'-X'FC'
Corea	949	X'00'-X'7F'	X'8F'-X'FE'
Taiwán	950	X'00'-X'7E'	X'81'-X'FE'
China	1381	X'00'-X'7F'	X'8C'-X'FE'
Corea	1363	X'00'-X'7F'	X'81'-X'FE'
China	1386	X'00'	X'81'-X'FE'
Nota: (*) esta es una página de códigos antigua que ya no se recomienda.			

Los puntos de código que no están asignados a ninguna de estas categorías no están definidos y se procesan como puntos de código no definidos de un solo byte.

Dentro de cada tabla de códigos DBCS implícita, hay 256 puntos de código disponibles como el segundo byte para cada primer byte válido. Los valores de segundo byte pueden adoptar cualquier valor comprendido entre X'40' y X'7E' y entre X'80' y X'FE'. Tenga en cuenta que, en entornos DBCS, DB2 no realiza la comprobación de validez de caracteres individuales de doble byte.

Juegos de caracteres de Extended UNIX Code (EUC)

Cada página de códigos EUC permite puntos de código de caracteres de un solo byte y un máximo de tres juegos diferentes de puntos de código de caracteres de varios bytes. Este soporte se consigue reservando un subconjunto de los 256 puntos de código disponibles de cada identificador de página de códigos SBCS relacionado para caracteres de un solo byte. El resto de los puntos de código son indefinidos, están asignados como un elemento de un carácter de varios bytes o están asignados como un introductor de un solo desplazamiento de un carácter de varios bytes. Estos puntos de código se muestran en las tablas siguientes.

Tabla 31. Puntos de código EUC japonés

Grupo	1er byte	2º byte	3er byte	4º byte
G0	X'20'-X'7E'	n/d	n/d	n/d
G1	X'A1'-X'FE'	X'A1'-X'FE'	n/d	n/d
G2	X'8E'	X'A1'-X'FE'	n/d	n/d
G3	X'8E'	X'A1'-X'FE'	X'A1'-X'FE'	n/d

Tabla 32. Puntos de código EUC coreano

Grupo	1er byte	2º byte	3er byte	4º byte
G0	X'20'-X'7E'	n/d	n/d	n/d
G1	X'A1'-X'FE'	X'A1'-X'FE'	n/d	n/d
G2	n/d	n/d	n/d	n/d
G3	n/d	n/d	n/d	n/d

Tabla 33. Puntos de código EUC chino tradicional

Grupo	1er byte	2º byte	3er byte	4º byte
G0	X'20'-X'7E'	n/d	n/d	n/d
G1	X'A1'-X'FE'	X'A1'-X'FE'	n/d	n/d
G2	X'8E'	X'A1'-X'FE'	X'A1'-X'FE'	X'A1'-X'FE'
G3	n/d	n/d	n/d	n/d

Tabla 34. Puntos de código EUC chino simplificado

Grupo	1er byte	2º byte	3er byte	4º byte
G0	X'20'-X'7E'	n/d	n/d	n/d
G1	X'A1'-X'FE'	X'A1'-X'FE'	n/d	n/d
G2	n/d	n/d	n/d	n/d

Tabla 34. Puntos de código EUC chino simplificado (continuación)

Grupo	1er byte	2º byte	3er byte	4º byte
G3	n/d	n/d	n/d	n/d

Los puntos de código que no están asignados a ninguna de estas categorías no están definidos y se procesan como puntos de código no definidos de un solo byte.

Programas CLI, ODBC, JDBC y SQLj en un entorno DBCS

Los programas JDBC y SQLj acceden a DB2 mediante el controlador CLI/ODBC de DB2 y por lo tanto utilizan el mismo archivo de configuración (db2cli.ini). Se deben añadir las siguientes entradas a este archivo de configuración si se ejecutan programas Java que acceden a DB2 Universal Database en un entorno DBCS:

PATCH1 = 65536
Impone que el controlador inserte manualmente una ″G″ delante de los literales de caracteres que de hecho sean literales gráficos. Este valor de PATCH1 se debe establecer siempre que se trabaje en un entorno de doble byte.

PATCH1 = 64
Impone que el controlador termine en NULL las series de salida gráficas. Es necesario para Microsoft Access en un entorno de doble byte. Si también tiene que utilizar este valor de PATCH1, deberá añadir los dos valores juntos (64+65536 = 65600) y establecer PATCH1=65600. Consulte la nota 2 en la página 444 siguiente para obtener más información sobre cómo especificar valores de PATCH1.

PATCH2 = 7
Impone que el controlador correlacione todos los tipos de datos de columnas de gráficos con el tipo de datos de columna de caracteres. Este valor de PATCH2 es necesario en un entorno de doble byte.

PATCH2 = 10
Sólo se debe utilizar en un entorno EUC (Extended Unix Code). Este valor de PATCH2 asegura que el controlador CLI proporciona datos correspondientes a variables de caracteres (CHAR, VARCHAR, etc.) en el formato adecuado para el controlador JDBC. Los datos de estos tipos de caracteres no se podrán utilizar en JDBC sin este valor.

Notas:

1. Cada una de estas palabras clave se establece en cada sección específica de base de datos del archivo db2cli.ini. Si desea establecerlas para varias bases de datos, repítalas para cada sección de base de datos de db2cli.ini.

2. Para establecer varios valores de PATCH1, añada los valores individuales y utilice la suma. Para establecer PATCH1 en 64 y 65536, establezca PATCH1=65600 (64+65536). Si ya tiene establecidos otros valores de PATCH1, sustituya el número existente por la suma de dicho número y los nuevos valores de PATCH1 que desee añadir.
3. Para establecer varios valores de PATCH2, especifíquelos en una serie delimitada por comas (a diferencia de la opción PATCH1). Para establecer los valores 1 y 7 para PATCH2, establezca PATCH2=″1,7″

Consideraciones sobre los juegos de códigos EUC y UCS-2 de japonés y chino tradicional

Las secciones siguientes describen las consideraciones a tener en cuenta para los juegos de códigos EUC y UCS-2 de japonés y chino tradicional.

Consideraciones sobre los juegos de códigos EUC y UCS-2 de japonés y chino tradicional

Extended UNIX Code (EUC) indica un conjunto de normas generales de codificación que pueden soportar entre uno y cuatro juegos de caracteres en los entornos operativos basados en UNIX. Las normas de codificación se basan en la definición ISO 2022 para la codificación de datos de 7 bits y de 8 bits, en los que se utilizan caracteres de control para separar algunos de los juegos de caracteres. Un juego de códigos basado en EUC se ajusta a las normas de codificación de EUC, pero también identifica los juegos de caracteres específicos asociados a las instancias concretas. Por ejemplo, el juego de códigos IBM-eucJP para japonés hace referencia a la codificación de caracteres en Japanese Industrial Standard según las normas de codificación de EUC.

El soporte de datos gráficos (caracteres puros de doble byte) por parte de la base de datos y de la aplicación cliente, mientras se ejecutan bajo páginas de códigos EUC con una codificación de caracteres de longitud superior a dos bytes, es limitado. Los productos DB2 Universal Database implementan normas estrictas para los datos gráficos que requieren que todos los caracteres contengan exactamente dos bytes. Estas normas no admiten muchos caracteres de las páginas de códigos EUC del japonés ni del chino tradicional. Para vencer esta situación, se proporciona soporte, tanto a nivel de aplicación como a nivel de base de datos, para representar los datos gráficos EUC en japonés y en chino tradicional utilizando otro esquema de codificación.

Una base de datos creada bajo EUC de japonés o de chino tradicional realmente almacenará y manipulará los datos gráficos utilizando el juego de códigos Unicode UCS-2, esquema de codificación de doble byte que constituye un subconjunto adecuado del repertorio completo de caracteres Unicode. De forma parecida, una aplicación que se ejecute bajo estas páginas

de códigos enviará los datos gráficos a la base de datos en forma de datos codificados en UCS-2. Mediante este soporte, las aplicaciones que se ejecutan bajo páginas de códigos EUC pueden acceder a los mismos tipos de datos que las que se ejecutan bajo páginas de códigos DBCS. El identificador de página de códigos definido por IBM asociado a UCS-2 es 1200, y el número de CCSID para la misma página de códigos es 13488. Los datos gráficos de una base de datos eucJP o eucTW utilizan el número de CCSID 13488. En una base de datos Unicode, utilice el CCSID 1200 para los datos GRAPHIC (gráficos).

DB2 Universal Database soporta todos los caracteres Unicode que se pueden codificar utilizando UCS-2, pero no realiza ninguna composición, descomposición ni normalización de los caracteres. Puede encontrar más información acerca del estándar Unicode en el sitio de Unicode Consortium, `www.unicode.org`, y en la última edición de la publicación sobre el estándar Unicode publicada por Addison Wesley Longman, Inc.

Si trabaja con aplicaciones o bases de datos que utilizan estos juegos de caracteres, es posible que tenga que considerar la posibilidad de utilizar datos codificados UCS-2. Cuando se convierten datos gráficos UCS-2 a la página de códigos EUC de la aplicación, existe la posibilidad de que se incremente la longitud de los datos. Si se visualizan grandes cantidades de datos, puede que sea necesario asignar almacenamientos intermedios, convertir y visualizar los datos en una serie de fragmentos.

En los apartados siguientes se explica cómo manejar los datos en este entorno. En estas secciones, el término EUC se utiliza únicamente para hacer referencia a los juegos de caracteres EUC del japonés y del chino tradicional. Observe que las explicaciones no se aplican al soporte por parte de DB2 de EUC del coreano y del chino simplificado, puesto que los datos gráficos de estos juegos de caracteres se representan utilizando la codificación EUC.

Conceptos relacionados:

- “Factor de expansión de conversión de página de códigos” en la página 440
- “Desbordamiento de longitud de serie de conversión de página de códigos en entornos de juegos de códigos mixtos” en la página 457

Consulta relacionada:

- “Supported territory codes and code pages” en el manual *Administration Guide: Planning*
- “Juegos de caracteres de Extended UNIX Code (EUC)” en la página 442

Consideraciones sobre las bases de datos y los clientes de doble byte con EUC mixtos

La administración de objetos de base de datos en entornos con páginas mixtas de códigos EUC y de doble byte resulta complicada por la posible expansión o contracción de la longitud de los nombres de objeto como consecuencia de conversiones entre la página de códigos del cliente y la de la base de datos. En concreto, muchos mandatos de administración y programas de utilidad tienen documentados límites en la longitud de las series de caracteres que pueden tomar como parámetros de entrada o salida. Estos límites se suelen imponer en el cliente, a no ser que se documente lo contrario. Por ejemplo, el límite para un nombre de tabla es de 128 bytes. Es posible que una serie de caracteres que contiene 128 bytes bajo una página de códigos de doble byte sea mayor, pongamos de 135 bytes, bajo una página de códigos EUC. Los mandatos tales como REORGANIZE TABLE no considerarían válido este hipotético nombre de tabla de 135 bytes si se utiliza como parámetro de entrada, a pesar de que sea válido en la base de datos de doble byte de destino. De forma parecida, la longitud máxima permitida para los parámetros de salida se puede exceder, después de una conversión de la página de códigos de la base de datos a la página de códigos de la aplicación. Esto puede ocasionar un error de conversión o que se produzca un truncamiento de los datos de salida.

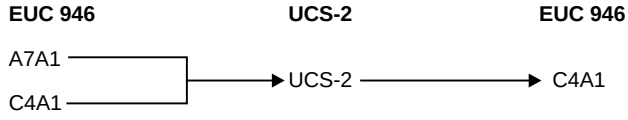
Si prevé que va a hacer un amplio uso de mandatos de administración y programas de utilidad en un entorno mixto con EUC y doble byte, debe definir los objetos de base de datos y los datos asociados a los mismos con la posibilidad de que la longitud supere los límites soportados. La administración de una base de datos EUC desde un cliente de doble byte impone menos restricciones que la administración de una base de datos de doble byte desde un cliente EUC. Normalmente, las series de caracteres de doble byte son iguales o más cortas que las series de caracteres EUC correspondientes. Esta característica generalmente conducirá a que se produzcan menos problemas debidos a la imposición de los límites en la longitud de las series de caracteres.

Nota: en el caso de las sentencias de SQL, la validación de los parámetros de entrada no se lleva a cabo hasta que se ha convertido la sentencia completa a la página de códigos de la base de datos. Así, se pueden utilizar series de caracteres que pueden ser técnicamente más largas de lo permitido cuando se representen en la página de códigos del cliente, pero que se ajusten a los requisitos de longitud cuando se representen en la página de códigos de la base de datos.

Consideraciones sobre la conversión de caracteres para usuarios de chino tradicional

Debido a la definición de estándares para el chino tradicional, existe un efecto colateral con que se puede encontrar cuando convierta algunos caracteres de páginas de códigos EUC o de doble byte y UCS-2. Existen 189 caracteres (que consisten en 187 radicales y 2 números) que comparten el mismo elemento de código UCS-2, cuando se convierten, que otro carácter del juego de códigos. Cuando estos caracteres se vuelven a convertir a EUC o doble byte, se convierten al elemento de código del mismo ideograma UCS-2, en lugar del elemento de código original. Cuando se visualizan, los caracteres parecen iguales, pero tienen un elemento de código distinto. Según el diseño de la aplicación, es posible que deba tener en cuenta este comportamiento.

Como ejemplo, considere lo que sucede con el elemento de código A7A1 en la página de códigos EUC 964, cuando se convierte a UCS-2 y luego se vuelve a convertir a la página de códigos original, EUC 946:



Así, los elementos de código originales A7A1 y C4A1 terminan siendo C4A1 después de la conversión.

Si necesita las tablas de conversión de páginas de códigos correspondientes a las páginas de códigos EUC EUC 946 (EUC de chino tradicional) o 950 (Big-5 de chino tradicional) y UCS-2, consulte el recurso en línea Product and Service Technical Library.

Datos gráficos en aplicaciones EUC en japonés o chino tradicional

La información siguiente describe consideraciones sobre el desarrollo de aplicaciones EUC para datos gráficos, que incluyen constantes gráficas, datos gráficos en UDF, procedimientos almacenados, archivos DBCLOB y clasificación:

- Constantes gráficas

  Las constantes gráficas, o literales, se clasifican realmente como datos mixtos de tipo carácter, puesto que forman parte de una sentencia de SQL. El servidor de base de datos convierte implícitamente a la codificación de gráficos cualquier constante gráfica de una sentencia de SQL procedente de un cliente EUC japonés o chino tradicional. En las aplicaciones SQL se pueden utilizar caracteres gráficos compuestos por caracteres codificados en EUC. Un servidor de base de datos EUC convertirá dichos literales al juego de códigos de la base de datos gráficos, que será UCS-2. Las constantes

gráficas procedentes de los clientes EUC nunca deben contener caracteres de una sola anchura, como por ejemplo los caracteres ASCII CS0 de 7 bits o los caracteres en japonés EUC CS2 (Katakana).

- UDF

  Se invoca a las UDF en el servidor de base de datos y están destinadas a tratar con datos codificados en el mismo juego de códigos que la base de datos. En el caso de las bases de datos que se ejecutan bajo los juegos de caracteres del japonés o del chino tradicional, los datos mixtos de tipo carácter se codifican utilizando el juego de códigos EUC bajo el que se crea la base de datos. Los datos gráficos se codifican utilizando UCS-2. Las UDF tienen que reconocer y manejar datos gráficos codificados con UCS-2.

  Por ejemplo, supongamos que crea una UDF denominada VARCHAR y la UDF convierte una serie gráfica en una serie de caracteres mixtos. Si la base de datos se crea bajo el juego de códigos EUC, la función VARCHAR tendrá que convertir una serie gráfica codificada como UCS-2 en una representación EUC.

- Procedimientos almacenados

  Un procedimiento almacenado que se ejecute bajo un juego de códigos EUC del japonés o del chino tradicional debe ser capaz de reconocer y manejar datos gráficos codificados utilizando UCS-2. Con estos juegos de códigos, los datos gráficos recibidos o devueltos a través del SQLDA de entrada/salida del procedimiento almacenado se codifican utilizando UCS-2.

- Archivos DBCLOB

  Las consideraciones importantes que deben tenerse en cuenta para archivos DBCLOB son:

  – Se supone que un archivo de datos DBCLOB está en la página de códigos EUC de la aplicación. Para los archivos DBCLOB en EUC, los datos se convierten a UCS-2 en el cliente cuando se produce una lectura, y de UCS-2 en el cliente cuando se trata de una grabación.

  – El número de bytes leídos o grabados en el servidor se devuelve en el campo de longitud de datos de la variable de referencia a archivo. El número de bytes se basa en el número de caracteres codificados en UCS-2 que se leen del archivo o que se graban en el mismo. El número de bytes que realmente se lee del archivo o que se graba en el mismo puede ser mayor que lo que el servidor graba en el campo de longitud de datos.

- Clasificación

  Los datos gráficos se clasifican en secuencia binaria. Los datos mixtos se clasifican en la secuencia de clasificación de la base de datos aplicada en cada byte. Debido a las posibles diferencias en la clasificación de caracteres en un juego de códigos EUC y en un juego de códigos DBCS para el mismo

país/región, se pueden obtener resultados distintos cuando se clasifican los mismos datos en una base de datos EUC y en una base de datos DBCS.

Consulta relacionada:

- "Función escalar GRAPHIC" en el manual *Consulta de SQL, Volumen 1*
- "SELECT sentencia" en el manual *Consulta de SQL, Volumen 2*
- "Series gráficas" en el manual *Consulta de SQL, Volumen 1*

Desarrollo de aplicaciones en situaciones de páginas de códigos distintas

En función de los esquemas de codificación de caracteres utilizados por la página de códigos de la aplicación y la página de códigos de la base de datos, se puede producir o no producir un cambio en la longitud de una serie cuando ésta se convierte de la página de códigos de origen a la página de códigos de destino. Un cambio en la longitud se suele asociar a conversiones entre páginas de códigos de múltiples bytes con distintos esquemas de codificación, por ejemplo DBCS y EUC.

Generalmente, un posible incremento de la longitud es más grave que una posible disminución de ésta, puesto que una sobreasignación de memoria resulta menos problemática que una subasignación. Hay que tratar por separado las consideraciones sobre las aplicaciones para enviar o recuperar datos en función del lugar en que se puede producir la posible expansión. También es importante observar las diferencias entre una situación de *mejor caso* y de *peor caso* cuando se indica una expansión o contracción de la longitud. Los valores positivos, que indican una posible expansión, darán el factor de multiplicación del *peor caso*. Por ejemplo, un valor de 2 para el campo SQLERRD(1) o SQLERRD(2) significa que se necesitará un almacenamiento máximo del doble de la longitud de la serie para manejar los datos después de la conversión. Éste es un indicador del *peor caso*. En este ejemplo, el *mejor caso* sería que, tras la conversión, la longitud no variara.

Los valores negativos para SQLERRD(1) o SQLERRD(2), que indican una posible contracción, también proporcionan el factor de expansión del *peor caso*. Por ejemplo, un valor de -1 significa que el almacenamiento máximo necesario es igual a la longitud de la serie antes de la conversión. Claro que es posible que se necesite menos almacenamiento, pero prácticamente esto tiene poca utilidad, a no ser que la aplicación receptora sepa por adelantado cómo están estructurados los datos de origen.

Para asegurarse de que siempre tendrá asignado suficiente almacenamiento para abarcar la máxima expansión posible después de una conversión de caracteres, debe asignar un almacenamiento igual al valor de `max_target_length` obtenido del cálculo siguiente:

1. Determine el factor de expansión de los datos.

   Para las transferencias de datos desde la aplicación a la base de datos:

   ```
   expansion_factor = ABS[SQLERRD(1)]
   if expansion_factor = 0
      expansion_factor = 1
   ```

   Para las transferencias de datos desde la base de datos a la aplicación:

   ```
   expansion_factor = ABS[SQLERRD(2)]
   if expansion_factor = 0
      expansion_factor = 1
   ```

   En los cálculos anteriores, ABS hace referencia al valor absoluto.

   La comprobación de `expansion_factor = 0` es necesaria, puesto que algunos productos DB2 Universal Database devuelven 0 en SQLERRD(1) y en SQLERRD(2). Estos servidores no soportan conversiones de página de códigos que tengan como resultado una expansión o una merma de los datos; esto se representa mediante un factor de expansión de 1.

2. Cálculo intermedio de la longitud.

   ```
   temp_target_length = actual_source_length * expansion_factor
   ```

3. Determine la longitud máxima para el tipo de datos de destino.

Tipo de datos de destino	Longitud máxima del tipo (type_maximum_lenght)
CHAR	254
VARCHAR	32.672
LONG VARCHAR	32.700
CLOB	2.147.483.647

4. Determine la longitud máxima de destino.

   ```
   1  if temp_target_length < actual_source_length
         max_target_length = type_maximum_length
      else
   2  if temp_target_length > type_maximum_length
         max_target_length = type_maximum_length
      else
   3  max_target_length = temp_target_length
   ```

   Todas las comprobaciones anteriores son necesarias para tener en cuenta los desbordamientos que se puedan producir durante el cálculo de la longitud. Las comprobaciones específicas son:

   1 Se produce un desbordamiento numérico durante el cálculo de `temp_target_length` en el paso 2.

Si el resultado de multiplicar dos valores positivos juntos es mayor que el valor máximo para el tipo de datos, el resultado se *reinicia* y se devuelve un valor menor que el mayor de los dos.

Por ejemplo, el valor máximo de un entero de 2 bytes con signo (que se utiliza para la longitud de los tipos de datos que no son CLOB) es 32.767. Si `actual_source_length` es 25.000 y el factor de expansión es 2, `temp_target_lenght` es, teóricamente, 50.000. Este valor es demasiado alto para el entero de 2 bytes con signo, por lo que se reinicia y se devuelve -15.536.

Para el tipo de datos CLOB, se utiliza para la longitud un entero de 4 bytes con signo. El valor máximo de un entero de 4 bytes con signo es 2.147.483.647.

2 `temp_target_length` es demasiado grande para el tipo de datos.

La longitud de un tipo de datos no puede superar los valores listados en el paso 3.

Si la conversión requiere más espacio del que hay disponible en el tipo de datos, puede existir la posibilidad de utilizar un tipo de datos más grande para que contenga el resultado. Por ejemplo, si un valor CHAR(250) requiere 500 bytes para contener la serie convertida, ésta no cabrá en un valor CHAR porque la longitud máxima es de 254 bytes. Sin embargo, existe la posibilidad de utilizar VARCHAR(500) para contener el resultado después de la conversión. Consulte el tema sobre desbordamiento de longitud de serie en conversión de página de códigos en entornos de juegos de códigos mixtos para obtener más información sobre lo que sucede cuando los datos convertidos sobrepasan el límite correspondiente a un tipo de datos.

3 `temp_target_lenght` es la longitud correcta para el resultado.

Utilizando los valores de SQLERRD(1) y SQLERRD(2) devueltos al conectar con la base de datos y los cálculos anteriores, puede determinar si existen posibilidades de que la longitud de una serie aumente o disminuya como consecuencia de la conversión de caracteres. En general, un valor de 0 ó 1 indica que no se producirá ninguna expansión; un valor superior a 1 indica una posible expansión de la longitud; un valor negativo indica una posible contracción. Observe que los valores de ‘0’ sólo provendrán de productos DB2 Universal Database de versiones anteriores. Asimismo, estos valores no están definidos para otros productos de servidor de base de datos. La tabla siguiente contiene los valores a esperar para diversas combinaciones de página de códigos de aplicación y página de códigos de base de datos cuando se utiliza DB2 Universal Database.

Tabla 35. Valores de SQLCA.SQLERRD en CONNECT

Página de códigos de la aplicación	Página de códigos de la base de datos	SQLERRD(1)	SQLERRD(2)
SBCS	SBCS	+1	+1
DBCS	DBCS	+1	+1
eucJP	eucJP	+1	+1
eucJP	DBCS	-1	+2
DBCS	eucJP	+2	-1
eucTW	eucTW	+1	+1
eucTW	DBCS	-1	+2
DBCS	eucTW	+2	-1
eucKR	eucKR	+1	+1
eucKR	DBCS	+1	+1
DBCS	eucKR	+1	+1
eucCN	eucCN	+1	+1
eucCN	DBCS	+1	+1
DBCS	eucCN	+1	+1

Si los valores SQLERRD(1) o SQLERRD(2) indican una expansión en el servidor de base de datos o en el cliente de aplicación, debe tener en cuenta lo siguiente:

- Expansión en el servidor de base de datos

  Si la entrada SQLERRD(1) indica una expansión en el servidor de base de datos, la aplicación debe contemplar la posibilidad de que los datos de tipo carácter dependientes de la longitud que sean válidos en el cliente no lo sean en el servidor de base de datos, después de que se hayan convertido. Por ejemplo, los productos DB2 requieren que los nombres de columna no tengan una longitud superior a 128 bytes. Es posible que una serie de caracteres que tenga una longitud de 128 bytes codificada bajo una página de códigos DBCS se expanda sobrepasando el límite de 128 bytes cuando se convierta a una página de códigos EUC. Esta posibilidad implica que pueden existir actividades que sean válidas cuando la página de códigos de la aplicación y la página de códigos de la base de datos sean iguales y no lo sean cuando sean distintas. Cuando diseñe bases de datos EUC y DBCS para situaciones de páginas de códigos distintas, proceda con precaución.
- Expansión en la aplicación

  Si la entrada SQLERRD(1) indica una expansión en la aplicación cliente, la aplicación debe contemplar la posibilidad de que los datos de tipo carácter dependientes de la longitud verán expandida su longitud una vez que se

hayan convertido. Por ejemplo, se recupera una fila con una columna CHAR(128). Si las páginas de códigos de la base de datos y de la aplicación son iguales, la longitud de los datos devueltos es de 128 bytes. Sin embargo, en una situación de páginas de códigos distintos, 128 bytes de datos codificados bajo una página de códigos DBCS se pueden expandir hasta sobrepasar 128 bytes cuando se conviertan a una página de códigos EUC. Así, es posible que haya que asignar almacenamiento adicional para recuperar la serie completa.

Conceptos relacionados:

- "Desbordamiento de longitud de serie de conversión de página de códigos en entornos de juegos de códigos mixtos" en la página 457

Validación de parámetros basada en cliente en un entorno de juegos de códigos mixtos

Un efecto colateral importante de una potencial expansión o contracción de los datos de tipo carácter entre el cliente y el servidor implica la validación de los datos que se pasen entre la aplicación cliente y el servidor de base de datos. En una situación de páginas de códigos desiguales, es posible que los datos que se determine que son válidos en el cliente sean realmente no válidos en el servidor de base de datos después de una conversión de páginas de códigos. Y, a la inversa, los datos que no son válidos en el cliente pueden serlo en el servidor de base de datos tras la conversión.

Existe la posibilidad de que cualquier aplicación de usuario final o biblioteca de API no sea capaz de manejar todas las posibilidades en una situación de páginas de códigos desiguales. Además, mientras que algunas validaciones de parámetros, como por ejemplo la longitud de las series, se realizan en el cliente para los mandatos y las API, las señales internas de las sentencias de SQL no se verifican hasta que se han convertido a la página de códigos de la base de datos. Esta verificación puede conducir a situaciones en que sea posible utilizar una sentencia de SQL en un entorno de páginas de códigos desiguales para acceder a un objeto de base de datos, como por ejemplo una tabla, pero que no sea posible acceder al mismo objeto utilizando una API o un mandato determinados.

Piense en una aplicación que devuelva datos contenidos en una tabla proporcionada por un usuario final y compruebe que el nombre de tabla no tiene una longitud superior a 128 bytes. Considere ahora los escenarios siguientes para esta aplicación:

1. Se crea una base de datos DBCS. Desde un cliente DBCS, se crea una tabla (t1) con un nombre de tabla que tiene una longitud de 128 bytes. El nombre de tabla incluye varios caracteres que constarían de más de dos bytes si se convirtiera la serie a EUC, lo que daría como resultado que la representación EUC del nombre de tabla tuviera una longitud total de 131

bytes. Puesto que no existe ninguna expansión para las conexiones de DBCS a DBCS, el nombre de tabla tendría 128 bytes en el entorno de la base de datos y la sentencia CREATE TABLE sería satisfactoria.

2. Un cliente EUC se conecta a la base de datos DBCS. Crea una tabla (t2) cuyo nombre de tabla tiene una longitud de 120 bytes cuando se codifica como EUC y de 100 bytes cuando se convierte a DBCS. El nombre de tabla en la base de datos DBCS tiene 100 bytes. La sentencia CREATE TABLE es satisfactoria.
3. El cliente EUC crea una tabla (t3) cuyo nombre de tabla contiene 64 caracteres EUC (131 bytes). Cuando se convierte este nombre a DBCS, su longitud se reduce hasta el límite de 128 bytes. La sentencia CREATE TABLE es satisfactoria.
4. El cliente EUC invoca a la aplicación para cada una de las tablas (t1, t2 y t3) de la base de datos DBCS, que da el resultado siguiente:

Tabla	Resultado
t1	La aplicación considera que el nombre de tabla no es válido porque tiene una longitud de 131 bytes.
t2	Visualiza resultados correctos.
t3	La aplicación considera que el nombre de tabla no es válido porque tiene una longitud de 131 bytes.

5. Se utiliza el cliente EUC para consultar la base de datos DBCS desde el CLP. Aunque el nombre de tabla tenga una longitud de 131 bytes en el cliente, las consultas resultan satisfactorias porque el nombre de tabla tiene una longitud de 128 bytes en el servidor.

Sentencia DESCRIBE en entornos de juegos de códigos mixtos

Una sentencia DESCRIBE realizada para una base de datos EUC devolverá información sobre columnas mixtas de caracteres y gráficos (GRAPHIC) en base a la definición de dichas columnas en la base de datos. Esta información se basa en la página de códigos del servidor antes de que se convierta a la página de códigos del cliente.

Cuando se ejecuta una sentencia DESCRIBE para un elemento de una lista de selección que se resuelve en el contexto de la aplicación (por ejemplo, `VALUES SUBSTR(?,1,2)`) para cualquier dato de tipo carácter o gráfico implicado, se debe evaluar el valor de SQLLEN devuelto, junto con la página de códigos devuelta. Si la página de códigos devuelta es la misma que la página de códigos de la aplicación, no se produce ninguna expansión. Si la página de códigos devuelta es la misma que la página de códigos de la base de datos, es posible que se produzca una expansión. Los elementos de lista de selección que son FOR BIT DATA (página de códigos 0), o si en la página de códigos de

la aplicación no se convierten al devolverlos a la aplicación, no se produce ninguna expansión ni contracción de la longitud informada.

Las consideraciones son distintas para una aplicación EUC que accede a una base de datos DBCS, en comparación con una aplicación DBCS que accede a una base de datos EUC:

- Aplicación EUC que accede a una base de datos DBCS

  Si la página de códigos de la aplicación es una página de códigos EUC, y si ésta emite una sentencia DESCRIBE para una base de datos que tiene una página de códigos DBCS, la información devuelta para las columnas CHAR y GRAPHIC se devuelve en el contexto de la base de datos. Por ejemplo, una columna CHAR(5) devuelta como parte de una sentencia DESCRIBE que tiene un valor de cinco en el campo SQLLEN. En el caso de datos que no son EUC, se asignan cinco bytes de almacenamiento cuando se captan los datos de esta columna. Con los datos EUC, el caso puede ser distinto. Si tiene lugar una conversión de página de códigos de DBCS a EUC, se puede producir un incremento de la longitud de los datos debido a la codificación distinta utilizada para los caracteres de las columnas CHAR. Por ejemplo, con el juego de caracteres del chino tradicional, el aumento máximo es el doble. Es decir, la longitud máxima de un carácter en la codificación DBCS es de dos bytes, que puede aumentar hasta una longitud máxima de cuatro bytes en EUC. Para el juego de códigos del japonés, el incremento máximo también es el doble. Sin embargo, observe que, mientras que la longitud máxima de un carácter en DBCS del japonés es de dos bytes, puede aumentar hasta un máximo de tres bytes en EUC del japonés. Aunque este incremento sólo parece corresponder a un factor de 1,5, los caracteres Katakana de un solo byte en DBCS del japonés tienen sólo una longitud de un byte, mientras que en el EUC del japonés tienen una longitud de dos bytes.

  Los cambios posibles en la longitud de los datos como consecuencia de conversiones de caracteres sólo se aplican a los datos mixtos de tipo carácter. La codificación de datos de caracteres gráficos siempre tiene la misma longitud, dos bytes, independientemente del esquema de codificación. Para evitar una pérdida de datos, es necesario evaluar si existe una situación de páginas de códigos desiguales, y si ésta se produce entre una aplicación EUC y una base de datos DBCS. Puede determinar la página de códigos de la base de datos y la de la aplicación mediante las señales de la SQLCA devuelta por una sentencia CONNECT. Si existe una situación de este tipo, es necesario que la aplicación asigne almacenamiento adicional para los datos mixtos de tipo carácter en base al factor de expansión máxima para ese esquema de codificación.

- Aplicación DBCS que accede a una base de datos EUC

  Si la página de códigos de la aplicación es una página de códigos DBCS y emite una sentencia DESCRIBE para una base de datos EUC, la situación es parecida a la que se produce cuando una aplicación EUC accede a una base

de datos DBCS. Sin embargo, en esta situación es posible que la aplicación necesite menos almacenamiento que el indicado por el valor del campo SQLLEN. El peor caso en esta situación es que todos los datos sean de un solo byte o de doble byte bajo EUC, lo cual significa que se requieren exactamente SQLLEN bytes en el esquema de codificación DBCS. En cualquier otra situación se necesitan menos de SQLLEN bytes, puesto que se requiere un máximo de dos bytes para almacenar cualquier carácter EUC.

Conceptos relacionados:

- "Derivación de valores de página de códigos" en la página 430
- "Factor de expansión de conversión de página de códigos" en la página 440
- "Desbordamiento de longitud de serie de conversión de página de códigos en entornos de juegos de códigos mixtos" en la página 457

Consulta relacionada:

- "DESCRIBE sentencia" en el manual *Consulta de SQL, Volumen 2*

Datos de longitud fija y de longitud variable en entornos de juegos de códigos mixtos

Debido a un posible cambio en la longitud de las series cuando se producen conversiones entre páginas de códigos DBCS y EUC, debe contemplar la posibilidad de no utilizar tipos de datos de longitud fija. Según si necesita o no que se realice espacios en blanco de relleno, debe considerar la posibilidad de cambiar el SQLTYPE de una serie de caracteres de longitud fija por una serie de caracteres de longitud variable después de ejecutar la sentencia DESCRIBE. Por ejemplo, si se informa de una conexión de EUC a DBCS de un factor máximo de expansión de dos para una columna CHAR(5), la aplicación debe asignar diez bytes.

Si el SQLTYPE es de longitud fija, la aplicación EUC recibirá la columna como una corriente de datos EUC convertida a partir de los datos DBCS (que, en sí misma, puede tener hasta cinco bytes de espacios en blanco de relleno) con un espacio en blanco de relleno adicional si la conversión de página de códigos no ocasiona que el elemento de datos crezca hasta su tamaño máximo. Si el SQLTYPE es de longitud variable, se mantiene el significado original del contenido de la columna CHAR(5); no obstante, los cinco bytes de origen pueden tener un destino de entre cinco y diez bytes. De forma parecida, en caso de una posible reducción de los datos (aplicación DBCS y base de datos EUC), debe considerar la posibilidad de trabajar con tipos de datos de longitud variable.

Una alternativa a la asignación de espacio adicional o a la gestión del tipo de datos consiste en seleccionar los datos por fragmentos. Por ejemplo, para

seleccionar el mismo VARCHAR(3000), que puede alcanzar 6000 bytes de longitud después de la conversión, puede realizar dos selecciones separadas, `SUBSTR(VC3000, 1, LENGTH(VC3000)/2)` y `SUBSTR(VC3000, (LENGTH(VC3000)/2)+1)`, en 2 áreas de aplicación VARCHAR(3000). Este método constituye la única solución posible cuando el tipo de datos ya no se puede gestionar. Por ejemplo, un CLOB codificado en la página de códigos DBCS del japonés con una longitud máxima de 2 gigabytes, posiblemente alcanzará dos veces su tamaño cuando se codifique en la página de códigos EUC del japonés. Esto significa que los datos se tendrán que dividir en fragmentos, puesto que no existe soporte para un tipo de datos cuya longitud supere 2 gigabytes.

Desbordamiento de longitud de serie de conversión de página de códigos en entornos de juegos de códigos mixtos

En entornos de páginas de códigos desiguales EUC y DBCS, después de que tenga lugar una conversión se pueden producir situaciones en que en una columna no haya suficiente espacio asignado para acomodar la serie entera. En este caso, la expansión máxima será dos veces la longitud de la serie en bytes. En los casos en que la expansión supera la capacidad de la columna, se devuelve SQLCODE -334 (SQLSTATE 22524).

Esto conduce a situaciones que pueden no resultar obvias de inmediato ni consideradas previamente:

- Una sentencia de SQL no puede tener una longitud superior a 32 765 bytes. Si la sentencia es suficientemente compleja o utiliza suficientes constantes o nombres de objetos que puedan estar sujetos a una expansión al convertirlos, se puede alcanzar este límite antes de lo esperado.
- Los identificadores de SQL tienen permitido expandirse al convertirlos hasta su longitud máxima, que es de ocho bytes para los identificadores cortos y de 128 bytes para los identificadores largos.
- Los identificadores de lenguaje principal tienen permitido expandirse al convertirlos hasta su longitud máxima, que es de 255 bytes.
- Cuando se convierten los campos de caracteres de la estructura SQLCA, sólo tienen permitido expandirse hasta su longitud máxima definida.

Cuando diseñe aplicaciones para entornos de juegos de códigos mixtos, debe consultar la documentación adecuada si se encuentra con una de las siguientes situaciones:

- Columnas de serie correspondientes en selecciones completas con operaciones de establecimiento (UNION, INTERSECT y EXCEPT)
- Operandos de concatenación
- Operandos de predicados (con la excepción de LIKE)
- Expresiones de resultados de una sentencia CASE
- Argumentos de la función escalar COALESCE (y VALUE)

- Valores de expresión de la lista IN de un predicado IN
- Expresiones correspondientes de una cláusula VALUES de varias filas

En estas situaciones, pueden tener lugar conversiones a la página de códigos de la aplicación en lugar de hacerlo a la página de códigos de la base de datos.

Otras situaciones que debe tener en cuenta son aquellas en las que la conversión de caracteres da como resultado una longitud de serie que sobrepasa el límite correspondiente al tipo de datos y conversiones de página de códigos en procedimientos almacenados:

- Conversión de caracteres sobrepasa un límite de tipo de datos

  En entornos de páginas de códigos desiguales EUC y DBCS, después de que tenga lugar una conversión se pueden producir situaciones en que la longitud de una serie de caracteres mixtos o gráfica supere la longitud máxima permitida para ese tipo de datos. Si la longitud de la serie, después de una expansión, supera el límite del tipo de datos, no se produce ninguna gestión del tipo. En cambio, se devuelve un mensaje de error que indica que se ha excedido la longitud máxima de expansión permitida. Es más probable que se produzca esta situación mientras se evalúan predicados que durante las inserciones. En las inserciones, la aplicación conoce más pronto la anchura de la columna y el factor de expansión máxima se puede tener en cuenta enseguida. En muchos casos, se puede evitar este efecto colateral de la conversión de caracteres cambiando el valor por un tipo de datos asociado cuya longitud máxima sea mayor. Por ejemplo, la longitud máxima de un valor CHAR es de 254 bytes, mientras que la longitud máxima de un VARCHAR es de 32 672 bytes. En los casos en que la expansión no supera la longitud máxima del tipo de datos, se devuelve SQLCODE -334 (SQLSTATE 22524).
- Conversión de página de códigos en un procedimiento almacenado

  Los datos de caracteres mixtos o gráficos especificados en variables de lenguaje principal y SQLDA en invocaciones a `sqleproc()` o a CALL de SQL se convierten en las situaciones en que las páginas de códigos de la aplicación y de la base de datos son distintas. En los casos en que se produce una expansión de la longitud de la serie como consecuencia de la conversión, se recibe un SQLCODE -334 (SQLSTATE 22524) si no hay suficiente espacio asignado para manejar la expansión. Así, cuando cree procedimientos almacenados, se debe asegurar de proporcionar suficiente espacio para una expansión potencial de las series. Debe utilizar tipos de datos de longitud variable con un espacio asignado suficiente que permita la expansión.

Consulta relacionada:

- "Función escalar COALESCE" en el manual *Consulta de SQL, Volumen 1*

- “Función escalar VALUE” en el manual *Consulta de SQL, Volumen 1*
- “Selección completa” en el manual *Consulta de SQL, Volumen 1*
- “VALUES sentencia” en el manual *Consulta de SQL, Volumen 2*
- “Sentencia CASE” en el manual *Consulta de SQL, Volumen 2*
- “Predicados” en el manual *Consulta de SQL, Volumen 1*

Aplicaciones conectadas a bases de datos Unicode

Las aplicaciones de cualquier entorno de página de códigos se pueden conectar a una base de datos Unicode. Para las aplicaciones que se conectan a una base de datos Unicode, el gestor de bases de datos convierte los datos de serie de caracteres entre la página de códigos de la aplicación y la página de códigos de la base de datos (UTF-8). Para una base de datos Unicode, los datos GRAPHIC están en orden UCS-2 big-endian. Sin embargo, cuando se utiliza el procesador de línea de mandatos para recuperar datos gráficos, los caracteres gráficos también se convierten a la página de códigos del cliente. Esta conversión permite que el procesador de línea de mandatos visualice caracteres gráficos en el font actual. Se puede producir una pérdida de datos siempre que el gestor de bases de datos convierta caracteres UCS-2 a la página de códigos de un cliente. Los caracteres que el gestor de bases de datos no puede convertir en caracteres válidos en la página de códigos del cliente se sustituyen por el carácter de sustitución por omisión de dicha página de códigos.

Nota: la información que se aplica a las aplicaciones en código mixto también se aplica a las aplicaciones que se conectan a bases de datos Unicode.

Cuando DB2 convierte caracteres de una página de códigos a UTF-8, el número total de bytes que representan los caracteres se puede expandir o reducir, según la página de códigos y los elementos de código de los caracteres. Los caracteres ASCII de 7 bits permanecen invariables en UTF-8, y cada carácter ASCII necesita un byte. Los caracteres que no son ASCII pasan a ocupar más de un byte cada uno. Para obtener más información sobre conversiones UTF-8, consulte los documentos estándar de Unicode.

Para las aplicaciones que se conectan a una base de datos Unicode, los datos GRAPHIC ya están en Unicode. Para las aplicaciones que se conectan a bases de datos DBCS, los datos GRAPHIC se convierten entre la página de códigos DBCS de la aplicación y la página de códigos DBCS de la base de datos. Las aplicaciones Unicode deben llevar a cabo las conversiones necesarias a Unicode y desde ellas mismas o deben definir la opción WCHARTYPE CONVERT y utilizar `wchar_t` para datos gráficos.

Conceptos relacionados:

- “Unicode handling of data types” en el manual *Administration Guide: Planning*
- “String comparisons in a Unicode database” en el manual *Administration Guide: Planning*
- “Variables gráficas del lenguaje principal en C y C++” en la página 191
- “Consideraciones sobre nombres de paquetes en entornos de páginas de códigos combinadas” en la página 435
- “Consideraciones sobre las bases de datos y los clientes de doble byte con EUC mixtos” en la página 446
- “Validación de parámetros basada en cliente en un entorno de juegos de códigos mixtos” en la página 453
- “Sentencia DESCRIBE en entornos de juegos de códigos mixtos” en la página 454
- “Datos de longitud fija y de longitud variable en entornos de juegos de códigos mixtos” en la página 456
- “Desbordamiento de longitud de serie de conversión de página de códigos en entornos de juegos de códigos mixtos” en la página 457

Capítulo 16. Gestión de transacciones

Unidad de trabajo remota

Una unidad de trabajo es una sola transacción lógica. Consta de una secuencia de sentencias de SQL en que todas las operaciones se realizan satisfactoriamente o la secuencia global se considera no satisfactoria.

Una unidad de trabajo remota permite que un usuario o programa de aplicación lea o actualice datos que se encuentran en una ubicación por unidad de trabajo. Soporta el acceso a una base de datos dentro de una unidad de trabajo. Mientras que un programa de aplicación puede acceder a varias bases de datos remotas, sólo puede acceder a una base de datos de una unidad de trabajo.

Una unidad de trabajo remota tiene las características siguientes:
- Se soportan varias peticiones por unidad de trabajo.
- Se soportan varios cursores por unidad de trabajo.
- Cada unidad de trabajo sólo puede acceder a una base de datos.
- El programa de aplicación confirma o retrotrae la unidad de trabajo. En determinadas condiciones de error, el servidor puede retrotraer la unidad de trabajo.

Consideraciones sobre la actualización múltiple

Las secciones siguientes describen las actualizaciones múltiples y cómo desarrollar aplicaciones que realizan actualizaciones múltiples.

Actualización múltiple

Una actualización múltiple, también conocida como *unidad de trabajo distribuida* (DUOW) y como *confirmación en dos fases*, es una función que

permite que las aplicaciones actualicen datos en varios servidores de bases de datos remotas con garantía de la integridad. Un buen ejemplo de actualización múltiple es una transacción bancaria que implica una transferencia monetaria de una cuenta a otra que se encuentra en otro servidor de base de datos. En una transacción de este tipo, es vital que las actualizaciones que implementan la operación de cargo en una cuenta no se confirmen a no ser que también se confirmen las actualizaciones necesarias para procesar cargos en la otra cuenta. Las consideraciones sobre actualizaciones múltiples se aplican cuando los datos que representan estas cuentas son gestionados por dos servidores de bases de datos distintos.

Puede utilizar la actualización múltiple para leer y actualizar varias bases de datos DB2 Universal Database en una unidad de trabajo. Si ha instalado DB2 Connect o utiliza la función de DB2 Connect que se suministra con DB2 Universal Database Enterprise Edition, también puede utilizar la actualización múltiple con servidores de bases de datos iSeries, de sistema principal o AS/400 como DB2 Universal Database para OS/390 y z/OS y DB2 UDB para AS/400. Se aplican ciertas restricciones cuando se utiliza DB2 Connect en una actualización múltiple con otros servidores de bases de datos.

Un gestor de transacciones coordina la confirmación entre varias bases de datos. Si se utiliza un entorno de supervisor del proceso de transacciones (TP) tal como TxSeries CICS, el supervisor TP utiliza su propio gestor de transacciones. De lo contrario, se utiliza el gestor de transacciones suministrado con DB2. Los sistemas operativos DB2 Universal Database para UNIX y Windows de 32 bits son gestores de recursos que cumplen con XA (arquitectura ampliada). Los sistemas principales y los servidores de bases de datos iSeries a los que se accede con DB2 Connect son gestores de recursos que se ajustan a XA. Observe también que el gestor de transacciones de DB2 Universal Database *no* es un gestor de transacciones que se ajuste a XA, lo cual significa que el gestor de transacciones sólo puede coordinar bases de datos DB2.

Conceptos relacionados:

- "X/Open distributed transaction processing model" en el manual *Administration Guide: Planning*
- "Multisite Updates" en el manual *DB2 Connect User's Guide*

Cuándo utilizar la actualización múltiple

Una actualización múltiple es de la máxima utilidad cuando se desea trabajar con dos o más bases de datos y mantener la integridad de los datos. Por ejemplo, si cada sucursal de un banco tiene su propia base de datos, una aplicación de transferencia monetaria podría hacer lo siguiente:

1. Conectar con la base de datos del remitente.

2. Leer el saldo de la cuenta del remitente y verificar que tiene suficiente dinero.
3. Reducir el saldo de la cuenta del remitente en la cantidad transferida.
4. Conectar con la base de datos del receptor.
5. Aumentar el saldo de la cuenta del receptor con la cantidad transferida.
6. Confirmar las bases de datos.

Al realizar la transferencia de fondos dentro de una unidad de trabajo, se asegurará de que se actualicen ambas bases de datos o no se actualice ninguna de ellas.

Sentencias de SQL en aplicaciones de actualización múltiple

La tabla siguiente le muestra cómo codificar sentencias de SQL para una actualización múltiple. La columna de la izquierda muestra sentencias de SQL que no utilizan la actualización múltiple; la de la derecha, muestra sentencias parecidas que contienen una actualización múltiple.

Tabla 36. Sentencias de RUOW y de SQL con una actualización múltiple

Sentencias de RUOW	Sentencias de actualización múltiple

CONNECT TO D1
  SELECT
  UPDATE
  COMMIT
 	
CONNECT TO D2
  INSERT
  COMMIT
 	
CONNECT TO D1
  SELECT
  COMMIT
CONNECT RESET
	CONNECT TO D1
	  SELECT
	  UPDATE
	
	CONNECT TO D2
	  INSERT
	  RELEASE CURRENT
	
	SET CONNECTION D1
	  SELECT
	  RELEASE D1
	COMMIT

Las sentencias de SQL de la columna de la izquierda sólo acceden a una base de datos por cada unidad de trabajo. Ésta es una aplicación de unidad de trabajo remota (RUOW).

Las sentencias de SQL de la columna de la derecha acceden a más de una base de datos en una unidad de trabajo. Ésta es una aplicación de actualización múltiple.

Algunas sentencias de SQL se codifican e interpretan de forma distinta en una aplicación de actualización múltiple:

- No es necesario confirmar ni retrotraer la unidad de trabajo actual antes de conectar con otra base de datos.

- Cuando se conecta con otra base de datos, no se desconecta la conexión actual. En cambio, se pone en estado *latente*. Si la sentencia CONNECT falla, la conexión actual no se ve afectada.
- No se puede conectar con la cláusula USER/USING si ya existe una conexión actual o latente con la base de datos.
- Puede utilizar la sentencia SET CONNECTION para hacer que una conexión latente pase a ser la conexión actual.

  Se puede lograr el mismo efecto emitiendo una sentencia CONNECT para la base de datos latente. Este método no está permitido si SQLRULES se establece en STD. Puede establecer el valor de SQLRULES utilizando una opción de precompilador, el mandato SET CLIENT o la API. El valor por omisión de SQLRULES (DB2) permite conmutar conexiones mediante la sentencia CONNECT.
- En una selección, la posición del cursor no se ve afectada si se conmuta a otra base de datos y luego se vuelve a la base de datos original.
- La sentencia CONNECT RESET no desconecta la conexión actual y no confirma implícitamente la unidad de trabajo actual. En cambio, esta sentencia es equivalente a una conexión explícita con la base de datos por omisión (en caso de que se haya definido). Si no se define una conexión implícita, se devuelve SQLCODE -1024 (SQLSTATE 08003).
- Puede utilizar la sentencia RELEASE para marcar una conexión de forma que se desconecte en la próxima COMMIT. La sentencia RELEASE CURRENT se aplica a la conexión actual, la sentencia RELEASE *conexión* se aplica a la conexión mencionada y la sentencia RELEASE ALL se aplica a todas las conexiones.

  Una conexión que está marcada para ser liberada se puede seguir utilizando hasta que se desactive en la próxima sentencia COMMIT. Una retrotracción no desactiva la conexión; este comportamiento permite efectuar un reintento con las conexiones establecidas. Utilice la sentencia DISCONNECT (o una opción de precompilador) para desactivar las conexiones después de una confirmación o retrotracción.
- La sentencia COMMIT confirma todas las bases de datos de la unidad de trabajo (actual o latente).
- La sentencia ROLLBACK retrotrae todas las bases de datos de la unidad de trabajo y cierra los cursores retenidos para todas las bases de datos, tanto si se accede a ellos en la unidad de trabajo como si no es así.
- Todas las conexiones (incluidas las latentes y las marcadas para su liberación) se desconectan cuando termina el proceso de la aplicación.
- Después de cualquier conexión satisfactoria (incluyendo una sentencia CONNECT sin opciones, que sólo consulta la conexión actual) se devolverá un número en los campos SQLERRD(3) y SQLERRD(4) de la SQLCA.

El campo SQLERRD(3) devuelve información sobre si la base de datos con la que se ha conectado se puede actualizar actualmente en una unidad de trabajo. Los valores posibles de este campo son:

1 Se puede actualizar.

2 Sólo es de lectura.

El campo SQLERRD(4) devuelve la información siguiente sobre las características actuales de la conexión:

0 No es aplicable. Este estado sólo es posible si se ejecuta desde un cliente de nivel inferior que utiliza la confirmación de una fase y es un actualizador.

1 Confirmación de una fase.

2 Confirmación de una fase (sólo lectura). Este estado sólo se aplica a servidores de bases de datos de sistema principal, AS/400 o iSeries a los que accede con DB2 Connect *sin* iniciar el gestor de puntos de sincronismo de DB2 Connect.

3 Confirmación de dos fases.

Si está escribiendo herramientas o programas de utilidad, es posible que desee emitir un mensaje a los usuarios en caso de que la conexión sólo sea de lectura.

Precompilación de aplicaciones de actualización múltiple

Cuando precompile una aplicación de actualización múltiple, debe establecer la conexión del CLP en una conexión de tipo 1; de lo contrario, recibirá un SQLCODE 30090 (SQLSTATE 25000) cuando intente precompilar la aplicación. Cuando se precompila una aplicación que utiliza actualizaciones múltiples, se utilizan las opciones de precompilador siguientes:

CONNECT (1 | 2)
Especifique CONNECT 2 para indicar que esta aplicación utiliza la sintaxis de SQL para las aplicaciones de actualización múltiple. El valor por omisión, CONNECT 1, significa que se aplican a la aplicación las reglas normales (RUOW) para la sintaxis de SQL.

SYNCPOINT (ONEPHASE | TWOPHASE | NONE)
Si especifica SYNCPOINT TWOPHASE y DB2 coordina la transacción, DB2 necesita una base de datos para mantener la información de estado de la transacción. Cuando despliegue la aplicación, debe definir esta base de datos configurando el parámetro de configuración del gestor de bases de datos *tm_database*.

SQLRULES (DB2 | STD)
Especifica si en las aplicaciones de actualización múltiple se deben utilizar las reglas de DB2 o las reglas estándar (STD) basadas en

ISO/ANSI SQL92. Las reglas de DB2 permiten emitir una sentencia CONNECT para una base de datos latente; las reglas STD no lo permiten.

DISCONNECT (EXPLICIT | CONDITIONAL | AUTOMATIC)
Especifica qué conexiones de base de datos se deben desconectar cuando se ejecute COMMIT: sólo las bases de datos que están marcadas para su liberación con una sentencia RELEASE (EXPLICIT), todas las bases de datos que no tienen cursores WITH HOLD abiertos (CONDITIONAL) o todas las conexiones (AUTOMATIC).

Las opciones de precompilador de la actualización múltiple entran en vigor cuando se realiza la primera conexión con una base de datos. Puede utilizar la API SET CLIENT para sustituir los valores de conexión cuando no exista ninguna conexión (antes de que se establezca alguna conexión o después de que se desconecten todas ellas). Puede utilizar la API QUERY CLIENT para consular los valores de conexión actuales del proceso de la aplicación.

Si un objeto al que se hace referencia en el programa de aplicación no existe, el enlazador falla. Existen tres maneras posibles de tratar las aplicaciones de actualización múltiple:

- Puede dividir la aplicación en varios archivos, cada uno de los cuales accederá a una sola base de datos. Luego prepare y enlace lógicamente cada archivo a la única base de datos a que accede.
- Puede asegurarse de que exista cada tabla en cada base de datos. Por ejemplo, en las sucursales de un banco pueden haber bases de datos cuyas tablas sean idénticas (a excepción de los datos).
- Puede utilizar únicamente SQL dinámico.

Conceptos relacionados:

- “Sentencias de SQL en aplicaciones de actualización múltiple” en la página 463

Consulta relacionada:

- “CONNECT (Tipo 1) sentencia” en el manual *Consulta de SQL, Volumen 2*
- “CONNECT (Tipo 2) sentencia” en el manual *Consulta de SQL, Volumen 2*
- “sqlesetc - Set Client” en el manual *Administrative API Reference*
- “sqleqryi - Query Client Information” en el manual *Administrative API Reference*
- “Mandato PRECOMPILE” en el manual *Consulta de mandatos*

Consideraciones sobre parámetros de configuración correspondientes a aplicaciones de actualización múltiple

Los parámetros de configuración siguientes afectan a las aplicaciones que realizan actualizaciones múltiples. A excepción de *locktimeout*, los parámetros de configuración son parámetros de configuración del gestor de bases de datos. *locktimeout* es un parámetro de configuración de la base de datos.

tm_database
: Especifica qué base de datos va a actuar como gestor de transacciones para las confirmaciones de dos fases.

resync_interval
: Especifica el número de segundos que el sistema esperará entre intentos para tratar de resincronizar una transacción dudosa. (Una transacción dudosa es una transacción que completa satisfactoriamente la primera fase de una confirmación en dos fases pero falla durante la segunda fase.)

locktimeout
: Especifica el número de segundos antes de que se exceda el tiempo de espera de un bloqueo y se retrotraiga la transacción actual para una base de datos determinada. La aplicación debe emitir una sentencia ROLLBACK explícita para retrotraer todas las bases de datos que participan en la actualización múltiple. *locktimeout* es un parámetro de configuración de la base de datos.

tp_mon_name
: Especifica el nombre del supervisor TP, si lo hay.

spm_resync_agent_limit
: Especifica el número de agentes simultáneos que pueden realizar operaciones de resincronización con el servidor de sistema principal, AS/400 o iSeries utilizando SNA.

spm_name
: - Si se utiliza el gestor de puntos de sincronismo con una conexión TCP/IP de confirmación en dos fases, *spm_name* debe ser un identificador exclusivo dentro de la red. Cuando se crea una instancia de DB2, DB2 deduce el valor por omisión de *spm_name* a partir del nombre de sistema principal TCP/IP. El usuario puede modificar este valor si no resulta aceptable en su entorno. Para la conectividad de TCP/IP con los servidores de bases de datos de sistema principal, el valor por omisión debe resultar aceptable. Para las conexiones SNA con servidores de bases de datos de sistema principal, AS/400 o iSeries, este valor debe coincidir con un perfil de LU SNA definido en el producto SNA.

- Si se utiliza el gestor de puntos de sincronismo con una conexión SNA de confirmación en dos fases, el nombre del gestor de puntos de sincronismo debe tener el valor LU_NAME que se utiliza para la confirmación en dos fases.
- Si se utiliza el gestor de puntos de sincronismo tanto para TCP/IP como para SNA, se debe utilizar el valor de LU_NAME que se utiliza para la confirmación en dos fases.

Nota: las actualizaciones múltiples en un entorno con servidores de bases de datos de sistema principal, AS/400 o iSeries pueden necesitar el gestor de puntos de sincronismo.

spm_log_size
El número de páginas de 4 kilobytes de cada archivo de registro primario y secundario utilizado por el gestor de puntos de sincronismo para registrar información sobre las conexiones, el estado de las conexiones actuales, etc.

Hay otras consideraciones que deben tenerse en cuenta si la aplicación realiza actualizaciones múltiples que coordina un gestor de transacciones XA con conexiones a una base de datos de sistema principal, AS/400 o iSeries.

Conceptos relacionados:

- "Multisite Updates" en el manual *DB2 Connect User's Guide*
- "Multisite update and sync point manager" en el manual *DB2 Connect User's Guide*

Tareas relacionadas:

- "Enabling Multisite Updates using the Control Center" en el manual *DB2 Connect User's Guide*

Consulta relacionada:

- "Sync Point Manager Log File Path configuration parameter - spm_log_path" en el manual *Administration Guide: Performance*
- "Transaction Resync Interval configuration parameter - resync_interval" en el manual *Administration Guide: Performance*
- "Transaction Manager Database Name configuration parameter - tm_database" en el manual *Administration Guide: Performance*
- "Transaction Processor Monitor Name configuration parameter - tp_mon_name" en el manual *Administration Guide: Performance*
- "Lock Timeout configuration parameter - locktimeout" en el manual *Administration Guide: Performance*
- "Sync Point Manager Name configuration parameter - spm_name" en el manual *Administration Guide: Performance*

- "Sync Point Manager Log File Size configuration parameter - spm_log_file_sz" en el manual *Administration Guide: Performance*
- "Sync Point Manager Resync Agent Limit configuration parameter - spm_max_resync" en el manual *Administration Guide: Performance*

Acceso a servidores de sistema principal, AS/400 o iSeries

Procedimiento:

Si desea crear aplicaciones que accedan a sistemas de bases de datos diferentes (o los actualicen), debe:

1. Utilizar sentencias de SQL y opciones de precompilación/vinculación que se soporten en todos los sistemas de bases de datos a los que vayan a acceder las aplicaciones. Por ejemplo, los procedimientos almacenados no se soportan en todas las plataformas.

   Para productos IBM, consulte la documentación de SQL **antes** de empezar a codificar.
2. Si es posible, hacer que las aplicaciones comprueben SQLSTATE en lugar de SQLCODE.

   Si las aplicaciones van a utilizar DB2 Connect y desea utilizar los SQLCODE, considere la posibilidad de utilizar el recurso de correlación que brinda DB2 Connect para correlacionar las conversiones de SQLCODE entre bases de datos distintas.
3. Pruebe la aplicación con las bases de datos de sistema principal, AS/400 o iSeries (tales como DB2 Universal Database para OS/390 y z/OS, OS/400 o DB2 para VSE y VM) que pretenda soportar.

Conceptos relacionados:

- "Aplicaciones en entornos de sistema principal o iSeries" en la página 527

Transacciones simultáneas

Las secciones siguientes describen las transacciones simultáneas y cómo evitar problemas con las mismas.

Transacciones simultáneas

Algunas veces, resulta de utilidad para una aplicación tener varias conexiones independientes denominadas *transacciones simultáneas*. Mediante la utilización de transacciones simultáneas, una aplicación puede conectar con varias bases de datos al mismo tiempo, y puede establecer varias conexiones distintas con la misma base de datos.

Las API de contexto que se utilizan para el acceso a bases de datos de varias hebras permiten que una aplicación utilice transacciones simultáneas. Cada contexto creado en una aplicación es independiente de los otros contextos. Esto significa que se crea un contexto, se conecta con una base de datos utilizando el contexto y se ejecutan sentencias de SQL para la base de datos sin que se vean afectadas por actividades de otros contextos, tales como la ejecución de sentencias COMMIT o ROLLBACK.

Por ejemplo, suponga que está creando una aplicación que permite que un usuario ejecute sentencias de SQL para una base de datos, y que mantiene un registro de las actividades realizadas en una segunda base de datos. Puesto que el registro se debe mantener actualizado, es necesario emitir una sentencia COMMIT después de cada actualización del mismo, pero no se desea que las sentencias de SQL del usuario se vean afectadas por las confirmaciones del registro. Ésta es una situación perfecta para las transacciones simultáneas. En la aplicación, cree dos contextos: uno conectará con la base de datos del usuario y se utilizará para todo el SQL del usuario; el otro conectará con la base de datos del registro y se utilizará para actualizar el registro. Mediante este diseño, cuando confirme un cambio efectuado en la base de datos del registro, no afectará a la unidad de trabajo actual del usuario.

Otro beneficio de las transacciones simultáneas es que, si se retrotrae el trabajo sobre los cursores de una conexión, esto no tiene ningún efecto sobre los cursores de otras conexiones. Después de la retrotracción de la primera conexión, en las otras conexiones se mantienen tanto el trabajo realizado como las posiciones de los cursores.

Conceptos relacionados:

- "Objetivo del acceso a base de datos de varias hebras" en la página 227

Problemas potenciales con transacciones simultáneas

Una aplicación que utiliza transacciones simultáneas se puede encontrar con algunos problemas que no pueden surgir cuando se escribe una aplicación que utiliza una sola conexión. Cuando escriba una aplicación con transacciones simultáneas, tenga cuidado con los aspectos siguientes:

- Dependencias de base de datos entre dos o más contextos.

  Cada contexto de una aplicación tiene sus propios recursos de base de datos, incluidos los bloqueos sobre objetos de base de datos. Estos conjuntos distintos de recursos posibilitan que dos contextos, si están accediendo al mismo objeto de base de datos, lleguen a un punto muerto. El gestor de bases de datos detectará el punto muerto, uno de los contextos recibirá SQLCODE -911 y se retrotraerá la unidad de trabajo del mismo.
- Dependencias de aplicaciones entre dos o más contextos.

La conmutación de contextos dentro de una sola hebra crea dependencias entre los contextos. Si los contextos también tienen dependencias de base de datos, es posible que se produzca un punto muerto. Puesto que algunas de las dependencias se producen fuera del gestor de bases de datos, no se detectará el punto muerto y se suspenderá la aplicación.

Como ejemplo de este tipo de problema, piense en la aplicación siguiente:

```
   contexto 1
   UPDATE TAB1 SET COL = :new_val

   contexto 2
   SELECT * FROM TAB1
   COMMIT

   contexto 1
COMMIT
```

Suponga que el primer contexto ejecuta satisfactoriamente la sentencia UPDATE. La actualización establece bloqueos sobre todas las filas de TAB1. Ahora, el contexto 2 intenta seleccionar todas las filas de TAB1. Puesto que los dos contextos son independientes, el contexto 2 espera a causa de los bloqueos mantenidos por el contexto 1. Sin embargo, el contexto 1 no puede liberar sus bloqueos hasta que termina la ejecución del contexto 2. La aplicación está ahora en un punto muerto, pero el gestor de bases de datos no sabe que el contexto está esperando al contexto 2, por lo que no impondrá que se retrotraiga uno de los contextos. La dependencia no resuelta deja la aplicación suspendida.

Conceptos relacionados:

- "Cómo evitar puntos muertos para transacciones simultáneas" en la página 471

Cómo evitar puntos muertos para transacciones simultáneas

Puesto que el gestor de bases de datos no puede detectar puntos muertos entre contextos, debe diseñar y codificar la aplicación de forma que impida (o, como mínimo, permita que se eviten) los puntos muertos. Supongamos que tenemos el siguiente ejemplo, que puede dar lugar a una situación de punto muerto:

```
contexto 1
UPDATE TAB1 SET COL = :new_val

contexto 2
SELECT * FROM TAB1
COMMIT

contexto 1
COMMIT
```

Suponga que el primer contexto ejecuta satisfactoriamente la sentencia UPDATE. La actualización establece bloqueos sobre todas las filas de TAB1. Ahora, el contexto 2 intenta seleccionar todas las filas de TAB1. Puesto que los dos contextos son independientes, el contexto 2 espera a causa de los bloqueos mantenidos por el contexto 1. Sin embargo, el contexto 1 no puede liberar sus bloqueos hasta que termina la ejecución del contexto 2. La aplicación está ahora en un punto muerto, pero el gestor de bases de datos no sabe que el contexto está esperando al contexto 2, por lo que no impondrá que se retrotraiga uno de los contextos. La dependencia no resuelta deja la aplicación suspendida.

Puede evitar el punto muerto del ejemplo de varias formas:

- Libere todos los bloqueos mantenidos antes de conmutar contextos.

  Cambie el código de forma que el contexto 1 realice su confirmación antes de conmutar al contexto 2.
- No acceda a un objeto determinado desde más de un contexto a la vez.

  Cambie el código de forma que tanto la actualización como la selección se realicen desde el mismo contexto.
- Establezca el parámetro de configuración de la base de datos *locktimeout* en un valor distinto de -1.

  Aunque un valor distinto de -1 no impedirá el punto muerto, permitirá que se reanude la ejecución. Eventualmente, se retrotraerá el contexto 2, puesto que no puede obtener el bloqueo solicitado. Cuando se retrotraiga el contexto 2, el contexto 1 podrá continuar la ejecución (con lo cual liberará los bloqueos) y el contexto 2 podrá reintentar su trabajo.

Aunque las técnicas para evitar puntos muertos se describen en términos del ejemplo, las puede aplicar a todas las aplicaciones que utilizan transacciones simultáneas.

Conceptos relacionados:

- “Problemas potenciales con transacciones simultáneas” en la página 470

Consulta relacionada:

- “Lock Timeout configuration parameter - locktimeout” en el manual *Administration Guide: Performance*

Consideraciones sobre la programación de interfaces XA de X/Open

La Interfaz XA de es un estándar abierto para coordinar cambios en varios recursos, mientras que se asegura la integridad de dichos cambios. Normalmente, los productos de software conocidos como *supervisores del proceso de transacciones* utilizan la interfaz XA y, puesto que DB2 soporta esta

interfaz, se puede acceder simultáneamente a una o más bases de datos DB2 como recursos en un entorno de este tipo.

DB2 requiere una consideración especial cuando funciona en un entorno de proceso de transacciones distribuidas (DTP) que utiliza la interfaz XA, puesto que se utiliza un modelo distinto para el proceso de transacciones, en comparación con las aplicaciones que se ejecutan de forma independiente de un supervisor TP. Las características de este modelo de proceso de transacciones son:

- Dentro de una transacción se pueden modificar varios tipos de recursos recuperables (como, por ejemplo, bases de datos DB2).
- Los recursos se actualizan utilizando una confirmación en dos fases para cerciorarse de la integridad de las transacciones que se ejecutan.
- Los programas de aplicación envían peticiones para confirmar o retrotraer una transacción al producto supervisor TP, en lugar de hacerlo a los gestores de los recursos. Por ejemplo, en un entorno CICS, una aplicación emitirá `EXEC CICS SYNCPOINT` para confirmar una transacción, y una emisión de `EXEC SQL COMMIT` a DB2 no sería válida y resultaría innecesaria.
- El supervisor TP filtra la autorización para ejecutar transacciones y el software relacionado, por lo que los gestores de recursos tales como DB2 tratan el supervisor TP como único usuario autorizado. Por ejemplo, cualquier uso de una transacción CICS debe ser autentificado por CICS, y se debe otorgar a CICS el privilegio de acceso a la base de datos, en lugar de al usuario final que invoca a la aplicación CICS.
- Normalmente, muchos programas (transacciones) se ponen en cola y se ejecutan en un servidor de base de datos (que aparece ante DB2 como un solo programa de aplicación de ejecución prolongada).

Debido a la naturaleza exclusiva de este entorno, DB2 tiene un comportamiento y unos requisitos especiales para las aplicaciones codificadas para que se ejecuten en él:

- Dentro de una unidad de trabajo se puede conectar con varias bases de datos, y actualizarlas, sin tener en cuenta las opciones de precompilador o los valores del cliente de la unidad de trabajo distribuida.
- La sentencia DISCONNECT no está permitida y, de intentarla, se rechazará con SQLCODE -30090 (SQLSTATE 25000).
- La sentencia RELEASE no recibe soporte y se rechazará con -30090.
- Las sentencias COMMIT y ROLLBACK no están permitidas dentro de los procedimientos almacenados a los que accede una transacción de supervisor TP.
- Cuando se inhabilitan explícitamente los flujos de confirmación en dos fases para una transacción (que se denomina transacción *LOCAL* en la terminología de la Interfaz XA), dentro de esta transacción sólo se puede

acceder a una base de datos. Esta base de datos no puede ser una base de datos de sistema principal, AS/400 ni iSeries a la que se acceda utilizando la conectividad SNA. Se soportan transacciones locales para DB2 para OS/390 Versión 5 que utilizan la conectividad TCP/IP.

- Las transacciones LOCALES deben emitir SQL COMMIT o SQL ROLLBACK al final de cada transacción; de lo contrario, se considerará que la transacción forma parte de la próxima transacción que se procese.
- La conmutación entre conexiones de base de datos simultáneas se realiza mediante el uso de SQL CONNECT o de SQL SET CONNECTION. La autorización utilizada para una conexión no se puede cambiar especificando un ID de usuario o una contraseña en la sentencia CONNECT.
- Si un objeto de base de datos, tal como una tabla, una vista o un índice, no se califica completamente en una sentencia de SQL dinámico, se calificará implícitamente con el ID de autentificación simple bajo el que se ejecute el supervisor TP, en lugar de hacerlo con el ID del usuario.
- Cualquier uso de las sentencias COMMIT o ROLLBACK de DB2 para transacciones que no sean LOCALES será rechazado. Se devolverán los códigos siguientes:
  - SQLCODE -925 (SQLSTATE 2D521) para una COMMIT estática
  - SQLCODE -926 (SQLSTATE 2D521) para una ROLLBACK estática
  - SQLCODE -426 (SQLSTATE 2D528) para una COMMIT dinámica
  - SQLCODE -427 (SQLSTATE 2D529) para una ROLLBACK dinámica
- Las peticiones de COMMIT o ROLLBACK por parte de la CLI también son rechazadas.
- Manejo de retrotracciones iniciadas por una base de datos:

  En un entorno DTP, si un RM ha iniciado una retrotracción (por ejemplo, debido a un error del sistema o a una situación de punto muerto) para terminar su propia ramificación de una transacción global, no debe procesar ninguna otra petición del mismo proceso de aplicación hasta que se produzca una petición de punto de sincronismo iniciada por el gestor de transacciones. Esto incluye los puntos muertos que se producen dentro de un procedimiento almacenado. Para el gestor de bases de datos, esto significa que se rechacen todas las peticiones de SQL posteriores con SQLCODE -918 (SQLSTATE 51021), para informar al usuario de que debe retrotraer la transacción global con el servicio de puntos de sincronismo del gestor de transacciones, por ejemplo utilizando el mandato CICS SYNCPOINT ROLLBACK en un entorno CICS. En cambio si, por algún motivo, se solicita al TM que confirme la transacción, el RM informará al TM sobre la retrotracción y hará que el TM retrotraiga de todas formas los otros RM.
- Cursores declarados WITH HOLD:

  Los cursores declarados WITH HOLD se soportan en entornos XA/DTP para supervisores del proceso de transacciones CICS.

En los casos en que no se soporten cursores declarados WITH HOLD, una sentencia OPEN será rechazada con SQLCODE -30090 (SQLSTATE 25000), código de razón 03.

Es responsabilidad de las transacciones asegurarse de que los cursores especificados como WITH HOLD se cierren explícitamente cuando ya no sean necesarios; de lo contrario, es posible que otras transacciones los hereden, lo cual ocasionará conflictos o un uso innecesario de recursos.

Si el supervisor TP da soporte a cursores HOLD, `xa_commit`, `xa_rollback` y `xa_prepare` se deben emitir en la misma conexión que la transacción global.

- Las sentencias que actualizan o cambian una base de datos no están permitidas para las bases de datos que no soportan flujos de peticiones de confirmación en dos fases. Por ejemplo, el acceso a servidores de bases de datos de sistema principal, AS/400 o iSeries en entornos en los que el nivel 2 del protocolo DRDA (DRDA2) no recibe soporte.
- Durante la ejecución, se puede determinar si una base de datos soporta actualizaciones en un entorno XA emitiendo una sentencia CONNECT. El tercer símbolo SQLERRD tendrá el valor 1 si la base de datos se puede actualizar; de lo contrario, tendrá el valor 2.
- Si están restringidas las actualizaciones, sólo se permitirán las sentencias de SQL siguientes:

```
CONNECT
DECLARE
DESCRIBE
EXECUTE IMMEDIATE (en que el primer símbolo o palabra clave es SET pero
                   no SET CONSTRAINTS)
OPEN CURSOR
FETCH CURSOR
CLOSE CURSOR
PREPARE (en que el primer símbolo o palabra clave no es un blanco ni
         un paréntesis izquierdo es SET (que no sea SET CONSTRAINTS,
         SELECT, WITH ni VALUES)
SELECT...INTO
VALUES...INTO
```

Cualquier otro intento será rechazado con SQLCODE -30090 (SQLSTATE 25000).

La sentencia PREPARE sólo se podrá utilizar para preparar sentencias SELECT. La sentencia EXECUTE IMMEDIATE también está permitida para ejecutar sentencias SET de SQL que no devuelvan ningún valor de salida, como por ejemplo la sentencia SET SQLID de DB2 Universal Database para OS/390 y z/OS.

- Restricciones de las API:

  Las API que emiten internamente una confirmación en la base de datos y eluden el proceso de confirmación en dos fases serán rechazadas con SQLCODE -30090 (SQLSTATE 25000). Para ver una lista de estas API,

consulte el apartado sobre restricciones en aplicaciones de actualización múltiple. Estas API no se soportan en una actualización múltiple (Connect Tipo 2).

- DB2 da soporte a un entorno XA / DTP de varias hebras.

Observe que las restricciones anteriores se aplican a las aplicaciones que se ejecutan en un entorno de supervisor TP que utiliza la interfaz XA. Si las bases de datos DB2 no están definidas para que se utilicen con la interfaz XA, no se aplican estas restricciones; no obstante, sigue siendo necesario asegurarse de que las transacciones se codifiquen de manera que no dejen DB2 en un estado que afecte adversamente a la próxima transacción que se vaya a ejecutar.

Conceptos relacionados:

- "Security considerations for XA transaction managers" en el manual *Administration Guide: Planning*
- "Configuration considerations for XA transaction managers" en el manual *Administration Guide: Planning*
- "XA function supported by DB2 UDB" en el manual *Administration Guide: Planning*
- "Actualización múltiple con DB2 Connect" en la página 539

Tareas relacionadas:

- "Updating host or iSeries database servers with an XA-compliant transaction manager" en el manual *Administration Guide: Planning*

Enlace de aplicaciones y la interfaz XA de X/Open

Para producir una aplicación ejecutable, es necesario que enlace los objetos de la aplicación con las bibliotecas de lenguajes, las bibliotecas del sistema operativo, las bibliotecas normales del gestor de bases de datos y las bibliotecas del supervisor TP y de los productos gestores de transacciones.

Capítulo 17. Consideraciones sobre programación para entornos de bases de datos particionadas

Cursores FOR READ ONLY en un entorno de bases de datos particionadas

Si declara un cursor desde el cual sólo pretende leer, incluya FOR READ ONLY o FOR FETCH ONLY en la declaración del OPEN CURSOR. (FOR READ ONLY y FOR FETCH ONLY son sentencias equivalentes.) Los cursores FOR READ ONLY permiten que la partición coordinadora recupere varias filas a la vez, mejorando en gran medida el rendimiento de las sentencias FETCH posteriores. Cuando los cursores no se declaran explícitamente como FOR READ ONLY, la partición coordinadora los trata como cursores actualizables. Los cursores actualizables incurren en un gasto considerable, puesto que requieren que la partición coordinadora sólo recupere una única fila por cada FETCH.

DDS dirigida y elusión local

Las secciones siguientes describen las consideraciones a tener en cuenta para utilizar DDS dirigida y elusión local en entornos de bases de datos particionadas.

DDS dirigida y elusión local en entornos de bases de datos particionadas

Para optimizar las aplicaciones de proceso de transacciones en línea (OLTP), es posible que desee evitar las sentencias simples de SQL que requieran un proceso en todas las particiones de bases de datos. Debe diseñar la aplicación

de forma que las sentencias de SQL puedan recuperar datos de particiones de bases de datos únicas. Las técnicas de subsección distribuida dirigida (DSS) y de elusión local evitan el gasto en que incurre la partición coordinadora al comunicar con una o con todas las particiones asociadas.

Conceptos relacionados:

- "DDS dirigida en entornos de bases de datos particionadas" en la página 478
- "Elusión local en entornos de bases de datos particionadas" en la página 479

DDS dirigida en entornos de bases de datos particionadas

Una subsección distribuida (DSS) es la acción de enviar subsecciones a la partición de base de datos que necesita realizar trabajo para una consulta paralela. También describe la iniciación de subsecciones mediante la invocación de valores específicos, como por ejemplo valores de variables en un entorno OLTP. Una *DDS dirigida* utiliza la clave de particionamiento de tabla para dirigir una consulta a una sola partición. Utilice este tipo de consulta en la aplicación para evitar la actividad general de la partición coordinadora necesaria para difundir una consulta a todas las particiones.

A continuación mostramos un fragmento de una sentencia SELECT de ejemplo que se puede aprovechar de las DSS dirigidas:

```
SELECT ... FROM t1
  WHERE PARTKEY=:hostvar
```

Cuando la partición coordinadora recibe la consulta, determina qué partición de base de datos contiene el subconjunto de datos para *:hostvar* y dirige la consulta, específicamente, a dicha partición de base de datos.

Para optimizar la aplicación utilizando DSS dirigidas, divida las consultas complejas en varias consultas simples. Por ejemplo, en la consulta siguiente la partición coordinadora compara la clave de particionamiento con varios valores. Dado que los datos que se ajustan a la consulta residen en varias particiones de base de datos, la partición coordinadora difunde la consulta a todas las particiones de base de datos:

```
SELECT ... FROM t1
  WHERE PARTKEY IN (:hostvar1, :hostvar2)
```

En lugar de esto, divida la consulta en varias sentencias SELECT (cada una de ellas con una sola variable del lenguaje principal) o utilice una única sentencia SELECT con UNION para lograr el mismo resultado. La partición coordinadora se puede aprovechar de sentencias SELECT más sencillas para utilizar DSS dirigidas a fin de comunicar únicamente con las particiones de base de datos necesarias. La consulta optimizada tiene el aspecto siguiente:

```
SELECT ... AS res1 FROM t1
  WHERE PARTKEY=:hostvar1
  UNION
SELECT ... AS res2 FROM t1
  WHERE PARTKEY=:hostvar2
```

Observe que la técnica anterior sólo mejorará el rendimiento si el número de selecciones de UNION es significativamente menor que el número de particiones.

Elusión local en entornos de bases de datos particionadas

Una forma especializada de la consulta DSS dirigida accede a los datos almacenados únicamente en la partición coordinadora. Esto se denomina *elusión local* porque la partición coordinadora completa la consulta sin tener que comunicar con otra partición.

Si es posible, la elusión local se habilita automáticamente, pero el usuario puede incrementar su uso dirigiendo transacciones a la partición de base de datos que contiene los datos correspondientes a dicha transacción. Una técnica para hacerlo consiste en hacer que un cliente remoto mantenga conexiones con cada una de las particiones de bases de datos. Luego una transacción puede utilizar la conexión correcta según la clave de particionamiento de entrada. Otra técnica consiste en agrupar las transacciones por particiones de bases de datos y disponer de un servidor de aplicaciones separado para cada partición de base de datos.

Para determinar el número de la partición de base de datos en la que residen los datos de la transacción, puede utilizar la API `sqlugrpn` (Obtener número de particionamiento de filas). Esta API permite que una aplicación calcule con eficacia el número de partición de una fila, dada la clave de particionamiento.

Otra alternativa consiste en utilizar el programa de utilidad `db2atld` para dividir los datos de entrada por número de partición y ejecutar una copia de la aplicación en cada partición de base de datos.

Consulta relacionada:

- “sqlugrpn - Get Row Partitioning Number” en el manual *Administrative API Reference*
- “db2atld - Mandato Cargador automático” en el manual *Consulta de mandatos*

Inserciones colocadas en almacenamiento intermedio

Las secciones siguientes describen las consideraciones a tener en cuenta para utilizar inserciones colocadas en almacenamiento intermedio en entornos de bases de datos particionadas.

Inserciones en almacenamiento intermedio en entornos de bases de datos particionadas

Una inserción en almacenamiento intermedio es una sentencia de inserción que se aprovecha de las colas de tablas para poner en el almacenamiento intermedio las filas que se insertan, obteniendo así una mejora significativa en el rendimiento. Para utilizar una inserción en almacenamiento intermedio, una aplicación se debe preparar o enlazar lógicamente con la opción INSERT BUF.

Las inserciones en almacenamiento intermedio pueden tener como consecuencia una mejora sustancial en el rendimiento de las aplicaciones que realizan inserciones. Normalmente, se puede utilizar una inserción en almacenamiento intermedio en las aplicaciones en que se utiliza una sola sentencia de inserción (y ninguna otra sentencia de modificación de bases de datos) dentro de un bucle para insertar muchas filas, y si el origen de los datos es una cláusula VALUES de la sentencia INSERT. Habitualmente, la sentencia INSERT hace referencia a una o más variables del lenguaje principal que cambian de valor durante las sucesivas ejecuciones del bucle. La cláusula VALUES puede especificar una sola fila o varias filas.

Las aplicaciones típicas de soporte de decisiones requieren una carga e inserción periódica de datos nuevos. Estos datos pueden representar cientos de miles de filas. Puede preparar y enlazar lógicamente las aplicaciones de forma que utilicen inserciones en almacenamiento intermedio cuando carguen tablas.

Para hacer que una aplicación utilice inserciones en almacenamiento intermedio, use el mandato PREP para procesar el archivo fuente del programa de aplicación, o utilice el mandato BIND en el archivo de vinculación resultante. En ambas situaciones, se debe especificar la opción INSERT BUF.

Nota: las inserciones en almacenamiento intermedio ocasionan que se lleven a cabo los pasos siguientes:

1. El gestor de bases de datos abre un almacenamiento intermedio de 4 KB para cada partición de base de datos en que reside la tabla.
2. La sentencia INSERT con la cláusula VALUES emitida por la aplicación hace que la o las fila(s) se coloque(n) en el o los almacenamiento(s) intermedio(s) apropiado(s).
3. El gestor de bases de datos devuelve el control a la aplicación.
4. Las filas del almacenamiento intermedio se envían a la partición cuando se llena el almacenamiento intermedio, o cuando se produce un suceso que hace que se envíen las filas de un almacenamiento intermedio parcialmente lleno. Un almacenamiento parcialmente lleno se vacía cuando se da una de las situaciones siguientes:

- La aplicación emite una sentencia COMMIT (implícita o explícitamente a través de la terminación de la aplicación) o ROLLBACK.
- La aplicación emite otra sentencia que hace que se tome un punto de rescate. Las sentencias de cursor OPEN, FETCH y CLOSE no ocasionan que se tome un punto de rescate ni cierran una inserción en almacenamiento intermedio abierta.

  Las sentencias de SQL siguientes cerrarán una inserción en almacenamiento intermedio abierta:
  - BEGIN COMPOUND SQL
  - COMMIT
  - DDL
  - DELETE
  - END COMPOUND SQL
  - EXECUTE IMMEDIATE
  - GRANT
  - INSERT en otra tabla
  - PREPARE de la misma sentencia dinámica (por nombres) realizando inserciones en almacenamiento intermedio
  - REDISTRIBUTE DATABASE PARTITION GROUP
  - RELEASE SAVEPOINT
  - REORG
  - REVOKE
  - ROLLBACK
  - ROLLBACK TO SAVEPOINT
  - RUNSTATS
  - SAVEPOINT
  - SELECT INTO
  - UPDATE
  - Ejecución de cualquier otra sentencia, pero no otra ejecución (bucle) de la INSERT en almacenamiento intermedio
  - Fin de la aplicación

  Las API siguientes cerrarán una inserción en almacenamiento intermedio abierta:
  - BIND (API)
  - REBIND (API)
  - RUNSTATS (API)
  - REORG (API)
  - REDISTRIBUTE (API)

En cualquiera de estas situaciones en que otra sentencia cierra la inserción en almacenamiento intermedio, la partición coordinadora espera hasta que todas las particiones de bases de datos reciben los almacenamientos intermedios y se insertan las filas. A continuación,

ejecuta la otra sentencia (aquélla que cierra la inserción en almacenamiento intermedio), siempre que todas las filas se hayan insertado satisfactoriamente.

La interfaz estándar en un entorno particionado (sin inserción en almacenamiento intermedio) carga una fila cada vez, llevando a cabo los pasos siguientes (suponiendo que la aplicación se ejecuta localmente en una de las particiones de bases de datos):

1. La partición coordinadora pasa la fila al gestor de bases de datos que se encuentra en la misma partición.
2. El gestor de bases de datos utiliza un procedimiento de hash indirecto para determinar la partición de base de datos en que se debe colocar la fila:
   - La partición de destino recibe la fila.
   - La partición de destino inserta la fila localmente.
   - La partición de destino envía una respuesta a la partición coordinadora.
3. La partición coordinadora recibe la respuesta de la partición de destino.
4. La partición coordinadora pasa la respuesta a la aplicación.

   No se confirma la inserción hasta que la aplicación emite una sentencia COMMIT.
5. Cualquier sentencia INSERT que contenga una cláusula VALUES es candidata a una inserción en almacenamiento intermedio, independientemente del número de filas o del tipo de elementos contenidos en las filas. Es decir, los elementos pueden ser constantes, registros especiales, variables del lenguaje principal, expresiones, funciones, etc.

Para una sentencia INSERT determinada con la cláusula VALUES, es posible que el compilador SQL de DB2 no coloque en el almacenamiento intermedio la inserción en base a consideraciones sobre la semántica, el rendimiento o la implementación. Si prepara o enlaza lógicamente su aplicación con la opción INSERT BUF, asegúrese de que no depende de una inserción en almacenamiento intermedio. Esto significa que:

- Se puede informar de errores de forma asíncrona para las inserciones en almacenamiento intermedio, o de forma síncrona para las inserciones normales. Si se informa de forma asíncrona, un error de inserción se puede notificar en una inserción posterior en el almacenamiento intermedio, o en la *otra* sentencia que cierre el almacenamiento intermedio. La sentencia que informa del error no se ejecuta. Por ejemplo, piense en la utilización de una sentencia COMMIT para cerrar un bucle de inserciones en almacenamiento intermedio. La confirmación informa de SQLCODE -803 (SQLSTATE 23505) debido a una clave duplicada procedente de una inserción anterior. En este escenario, no se ejecuta la confirmación. Si desea que la aplicación efectúe

realmente la confirmación, por ejemplo, algunas actualizaciones realizadas antes de entrar en el bucle de inserciones en almacenamiento intermedio, debe volver a emitir la sentencia COMMIT.

- Las filas insertadas pueden resultar visibles inmediatamente mediante una sentencia SELECT que utilice un cursor sin inserción en almacenamiento intermedio. Con una inserción en almacenamiento intermedio, las filas no serán visibles de inmediato. Si precompila o enlaza lógicamente la aplicación con la opción INSERT BUF, no la escriba de forma que dependa de estas filas seleccionadas por el cursor.

Las inserciones en almacenamiento intermedio tienen como consecuencia las ventajas siguientes:

- Sólo se envía un mensaje desde la partición de destino a la partición coordinadora por cada almacenamiento intermedio recibido por la partición de destino.
- Un almacenamiento intermedio puede contener un elevado número de filas, especialmente si las filas son pequeñas.
- Se produce un proceso paralelo a medida que se realizan inserciones en las particiones, mientras la partición coordinadora está recibiendo filas nuevas.

Una aplicación que se enlaza lógicamente con INSERT BUF se debe escribir de forma que la misma sentencia INSERT con la cláusula VALUES se reitere repetidamente antes de que se emita cualquier sentencia o API que cierre una inserción en almacenamiento intermedio.

Nota: para impedir que las inserciones en almacenamiento intermedio llenen el registro de transacciones, debe realizar confirmaciones periódicas.

Conceptos relacionados:

- “Creación y preparación del archivo fuente” en la página 79
- “Creación de paquetes mediante el mandato BIND” en la página 89
- “Consideraciones sobre la utilización de inserciones en almacenamiento intermedio” en la página 483
- “Restricciones en la utilización de inserciones en almacenamiento intermedio” en la página 486

Consideraciones sobre la utilización de inserciones en almacenamiento intermedio

Las inserciones en almacenamiento intermedio presentan comportamientos que pueden afectar a un programa de aplicación. Este comportamiento es debido a la naturaleza asíncrona de las inserciones en almacenamiento intermedio. En base a los valores de la clave de particionamiento de la fila, cada fila insertada se coloca en un almacenamiento intermedio destinado a la

partición correcta. Estos almacenamientos intermedios se envían a sus particiones de destino cuando se llenan o cuando un suceso ocasiona que se vacíen. Cuando diseñe y codifique la aplicación, debe tener en cuenta los aspectos siguientes:

- Existen determinadas condiciones de error de las que no se informa cuando se ejecuta la sentencia INSERT. Se informa de ellas más tarde, cuando se ejecuta la primera sentencia que no sea INSERT (o INSERT en otro tabla), como por ejemplo DELETE, UPDATE, COMMIT o ROLLBACK. Cualquier sentencia o API que cierre la sentencia de inserción en almacenamiento intermedio puede ver el informe de errores. Asimismo, cualquier invocación de la propia inserción puede ver un error de una fila insertada previamente. Por otra parte, si otra sentencia informa de un error de una inserción en almacenamiento intermedio, como por ejemplo UPDATE o COMMIT, DB2 no intentará ejecutar dicha sentencia.
- Un error detectado durante la inserción de un *grupo de filas* hace que se retrotraigan todas las filas de dicho grupo. Un grupo de filas se define como todas las filas insertadas mediante ejecuciones de una sentencia de inserción en almacenamiento intermedio:
  - Desde el principio de la unidad de trabajo,
  - Desde que se preparó la sentencia (si ésta es dinámica), o
  - Desde la ejecución anterior de otra sentencia de actualización. Para ver una lista de las sentencias que cierran (o vacían) una inserción en almacenamiento intermedio, consulte la descripción de inserciones en almacenamiento intermedio en entornos de bases de datos particionadas.
- Es posible que una fila insertada no resulte visible inmediatamente para la sentencias SELECT emitidas después de INSERT por el mismo programa de aplicación, en caso de que SELECT se ejecute utilizando un cursor.

Una sentencia INSERT en almacenamiento intermedio puede estar abierta o cerrada. La primera invocación de la sentencia abre la INSERT en almacenamiento intermedio, se añade la fila al almacenamiento intermedio apropiado y se devuelve el control a la aplicación. Las invocaciones posteriores añaden filas al almacenamiento intermedio, dejando la sentencia abierta. Mientras la sentencia está abierta, los almacenamientos intermedios se pueden enviar a sus particiones de destino, donde se insertan las filas en la partición de la tabla de destino. Si se invoca a cualquier sentencia o API que cierra una inserción en almacenamiento intermedio mientras hay una sentencia INSERT en almacenamiento intermedio abierta (incluida una invocación de *otra* sentencia INSERT en almacenamiento intermedio), o si se emite una sentencia PREPARE para una sentencia INSERT en almacenamiento intermedio abierta, antes de procesar la nueva petición se cierra la sentencia abierta. Si se cierra la sentencia INSERT en almacenamiento intermedio, se vacían los almacenamientos intermedios restantes. Entonces se envían las filas

a las particiones de destino y se insertan. El proceso de la nueva petición sólo comienza después de que se envíen todos los almacenamientos intermedios y se inserten todas las filas.

Si se detectan errores durante el cierre de la sentencia INSERT, la SQLCA de la nueva petición se llenará con la descripción del error y no se ejecutará la petición. Asimismo, se eliminará de la base de datos el grupo entero de filas que se hubieran insertado mediante la sentencia INSERT en almacenamiento intermedio *desde que se abrió*. El estado de la aplicación será tal como se haya definido para el error concreto que se haya detectado. Por ejemplo:

- Si el error es un punto muerto, se retrotrae la transacción (incluidos los cambios efectuados antes de que se abriera la sección de inserción en almacenamiento intermedio).
- Si el error es una violación de una clave exclusiva, el estado de la base de datos es igual que antes de que se abriera la sentencia. La transacción permanece activa y los cambios efectuados antes de que se abriera la sentencia no se ven afectados.

Por ejemplo, considere la aplicación siguiente que está enlazada lógicamente con la opción de inserción en almacenamiento intermedio:

```
EXEC SQL UPDATE t1 SET COMMENT='about to start inserts';
DO UNTIL EOF OR SQLCODE < 0;
  READ VALUE OF hv1 FROM A FILE;
  EXEC SQL INSERT INTO t2 VALUES (:hv1);
  IF 1000 INSERTS DONE, THEN DO
     EXEC SQL INSERT INTO t3 VALUES ('another 1000 done');
     RESET COUNTER;
  END;
END;
EXEC SQL COMMIT;
```

Suponga que el archivo contiene 8 000 valores, pero el valor 3 258 no es lícito (por ejemplo, contiene una violación de la clave exclusiva). Cada 1 000 inserciones dan como resultado la ejecución de otra sentencia de SQL que, a continuación, cierra la sentencia `INSERT INTO t2`. Durante el cuarto grupo de 1 000 inserciones, se detectará el error del valor 3 258. Se puede detectar después de la inserción de más valores (y no necesariamente después del siguiente). En esta situación, se devuelve un código de error para la sentencia `INSERT INTO t2`.

También es posible que se detecte el error cuando se intente una inserción en la tabla `t3`, la cual cerrará la sentencia `INSERT INTO t2`. En esta situación, se devuelve el código de error para la sentencia `INSERT INTO t3`, aunque el error corresponda a la tabla `t2`.

En cambio, suponga que tiene que insertar 3 900 filas. Antes de que se le indique el error de la fila número 3 258, la aplicación puede salir del bucle e

intentar emitir una sentencia COMMIT. El código de retorno de la violación de la clave exclusiva se emitirá para la sentencia COMMIT y no se ejecutará COMMIT. Si la aplicación desea confirmar (COMMIT) las 3000 filas que se encuentran hasta ahora en la base de datos (la última ejecución de `EXEC SQL INSERT INTO t3 ...` finaliza el punto de rescate para estas 3000 filas), habrá que *volver a emitir* la sentencia COMMIT. También se aplican consideraciones parecidas a ROLLBACK.

Nota: cuando utilice inserciones en almacenamiento intermedio, debe supervisar cuidadosamente los SQLCODE devueltos para evitar que una tabla quede en un estado indeterminado. Por ejemplo, si eliminara la cláusula `SQLCODE < 0` de la sentencia THEN DO del ejemplo anterior, la tabla podría terminar por contener un número indeterminado de filas.

Conceptos relacionados:

- "Inserciones en almacenamiento intermedio en entornos de bases de datos particionadas" en la página 480

Restricciones en la utilización de inserciones en almacenamiento intermedio

Se aplican las restricciones siguientes a las inserciones en almacenamiento intermedio:

- Para que una aplicación se aproveche de las inserciones en almacenamiento intermedio, se tiene que cumplir una de las condiciones siguientes:
  - La aplicación debe estar preparada mediante PREP o enlazada lógicamente con el mandato BIND y se debe especificar la opción INSERT BUF.
  - La aplicación debe estar enlazada lógicamente utilizando las API BIND o PREP con la opción SQL_INSERT_BUF.
- Si la sentencia INSERT con la cláusula VALUES incluye campos largos o LOB en la lista de columnas explícita o implícita, se pasa por alto la opción INSERT BUF para esta sentencia y se realiza una inserción normal, no en el almacenamiento intermedio. Ésta no es una condición de error y no se emite ningún mensaje de error ni de aviso.
- Una INSERT con una selección completa no se ve afectada por INSERT BUF. Una inserción en almacenamiento intermedio no mejora el rendimiento de este tipo de INSERT.
- Las inserciones en almacenamiento intermedio sólo se pueden utilizar en aplicaciones y no mediante inserciones emitidas por el CLP, puesto que éstas se realizan a través de la sentencia EXECUTE IMMEDIATE.

Luego, la aplicación se puede ejecutar desde cualquier plataforma cliente soportada.

Ejemplo de extracción un gran volumen de datos en una base de datos particionada

Aunque DB2 Universal Database proporciona características excelentes para el proceso de consultas paralelas, el único punto de conexión de una aplicación o un mandato EXPORT puede convertirse en un embotellamiento si se extraen grandes volúmenes de datos. Este cuello de botella se produce porque el proceso de pasar los datos del gestor de bases de datos a la aplicación hace un gran uso de la CPU que se ejecuta en una sola partición (normalmente, también en un solo procesador).

DB2 Universal Database brinda varios métodos para vencer el embotellamiento, de forma que el volumen de datos extraídos se escala linealmente por unidad de tiempo con un número de procesadores incrementado. El ejemplo siguiente describe la idea básica que se encuentra detrás de estos métodos.

Suponga que tiene una tabla llamada EMPLOYEE que está almacenada en 20 particiones de bases de datos, y que genera una lista de correo (FIRSTNME, LASTNAME, JOB) de todos los empleados que pertenecen a un departamento legítimo (es decir, en que WORKDEPT no es NULL).

La consulta siguiente se ejecuta en cada partición y luego genera el conjunto completo de respuestas en una sola partición (la partición coordinadora):

```
SELECT FIRSTNME, LASTNAME, JOB FROM EMPLOYEE WHERE WORKDEPT IS NOT NULL
```

Pero la consulta siguiente se puede ejecutar en cada partición correspondiente a la base de datos (es decir, si existen cinco particiones, se necesitan cinco consultas separadas, una en cada partición). Cada consulta genera el conjunto de todos los nombres de empleado cuyo registro se encuentra en la partición concreta en que se ejecuta la consulta. Cada conjunto de resultados local se puede redirigir a un archivo. Luego es necesario fusionar los conjuntos de resultados en un solo conjunto.

En AIX, puede utilizar una propiedad de los archivos del Sistema de archivos de red (NFS) para automatizar la fusión. Si todas las particiones dirigen sus conjuntos de respuestas al mismo archivo de un montaje por NFS, los resultados se fusionan. Observe que una utilización de NFS sin bloquear la respuesta en almacenamientos intermedios grandes tiene como consecuencia un rendimiento muy pobre.

```
SELECT FIRSTNME, LASTNAME, JOB FROM EMPLOYEE WHERE WORKDEPT IS NOT NULL
              AND NODENUMBER(NAME) = CURRENT NODE
```

El resultado se puede almacenar en un archivo local (lo que significa que el resultado final constará de 20 archivos, cada uno de los cuales contendrá una porción del conjunto de respuestas completo), o en un solo archivo montado por NFS.

En el ejemplo siguiente se utiliza el segundo método, de forma que el resultado se encuentra en un único archivo montado por NFS para los 20 nodos. El mecanismo de bloqueo de NFS asegura la colocación en serie de las grabaciones en el archivo de resultados por parte de las distintas particiones. Observe que este ejemplo, tal como se presenta, se ejecuta en la plataforma AIX con un sistema de archivos NFS instalado.

```
#define _POSIX_SOURCE
#define INCL_32

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sqlenv.h>
#include <errno.h>
#include <sys/access.h>
#include <sys/flock.h>
#include <unistd.h>

#define BUF_SIZE 1500000  /* Almacenamiento intermedio local para almacenar
                             los registros captados */
#define MAX_RECORD_SIZE 80 /* >= tamaño de un registro grabado */

int main(int argc, char *argv[]) {

    EXEC SQL INCLUDE SQLCA;
    EXEC SQL BEGIN DECLARE SECTION;
       char dbname[10];  /* Nombre de base de datos (argumento
                            del programa) */
       char userid[9];
       char passwd[19];
       char first_name[21];
       char last_name[21];
       char job_code[11];
    EXEC SQL END DECLARE SECTION;

       struct flock unlock ;  /* estructuras y variables para manejar */
       struct flock lock ;  /* el mecanismo de bloqueo NFS */
       int lock_command ;
       int lock_rc ;
       int iFileHandle ;  /* archivo de salida */
       int iOpenOptions = 0 ;
       int iPermissions ;
       char * file_buf ;  /* puntero al almacenamiento intermedio
                             en que se acumulan los registros captados */
       char * write_ptr ;  /* posición en que se graba el siguiente
                              registro */
       int buffer_len = 0 ;  /* longitud de la porción utilizada del
```

```
                              almacenamiento intermedio */

  /* Inicialización */

     lock.l_type = F_WRLCK;  /* Petición de bloqueo exclusivo de grabación */
     lock.l_start = 0;  /* Para bloquear el archivo entero */
     lock.l_whence = SEEK_SET;
     lock.l_len = 0;
     unlock.l_type = F_UNLCK;  /* Petición de liberación de un bloqueo */
     unlock.l_start = 0;  /* Para desbloquear el archivo entero */
     unlock.l_whence = SEEK_SET;
     unlock.l_len = 0;
     lock_command = F_SETLKW;  /* Establecer el bloqueo */
     iOpenOptions = O_CREAT;  /* Crear el archivo si no existe */
     iOpenOptions |= O_WRONLY;  /* Abrir sólo para grabación */

  /* Conectar con la base de datos */

     if (argc == 3) {
        strcpy( dbname, argv[2] ); /* obtener nombre de base de datos del argumento */
        EXEC SQL CONNECT TO :dbname IN SHARE MODE ;
  if ( SQLCODE != 0 ) {
           printf( "Error: CONNECT TO the database failed. SQLCODE = %ld\n",
           SQLCODE );
     exit(1);
 }
     }
     else if ( argc == 5 ) {
        strcpy( dbname, argv[2] ); /* obtener nombre de base de datos del argumento */
strcpy (userid, argv[3]);
strcpy (passwd, argv[4]);
EXEC SQL CONNECT TO :dbname IN SHARE MODE USER :userid USING :passwd;
if ( SQLCODE != 0 ) {
   printf( "Error: CONNECT TO the database failed. SQLCODE = %ld\n",
           SQLCODE );
     exit( 1 );
       }
     }
                  else
  printf ("\nUSAGE: largevol txt_file database [userid passwd]\n\n");
  exit( 1 ) ;
     } /* endif */

    /* Abrir el archivo de entrada con los permisos de acceso especificados */

   if ( ( iFileHandle = open(argv[1], iOpenOptions, 0666 ) ) == -1 ) {
     printf( "Error: Could not open %s.\n", argv[2] ) ;
     exit( 2 ) ;
  }

  /* Establecer escapes por error y fin de tabla */

  EXEC SQL WHENEVER SQLERROR GO TO ext ;
  EXEC SQL WHENEVER NOT FOUND GO TO cls ;
```

```
  /* Declarar y abrir el cursor */

  EXEC SQL DECLARE c1 CURSOR FOR
           SELECT firstnme, lastname, job FROM employee
           WHERE workdept IS NOT NULL
           AND NODENUMBER(lastname) = CURRENT NODE;
  EXEC SQL OPEN c1 ;

  /* Establecer almacenamiento intermedio temporal para almacenar el
                            resultado captado */

  if ( ( file_buf = ( char * ) malloc( BUF_SIZE ) ) == NULL ) {
     printf( "Error: Allocation of buffer failed.\n" ) ;
     exit( 3 ) ;
  }
  memset( file_buf, 0, BUF_SIZE ) ; /* restablecer almacenamiento intermedio */
  buffer_len = 0 ;  /* restablecer longitud del almacenamiento intermedio */
  write_ptr = file_buf ;  /* restablecer el puntero de grabación */
  /* Para cada registro captado realizar lo siguiente */
  /*  - insertarlo en el almac. interm. tras el       */
  /*    registro almacenado previamente               */
  /*  - comprobar si aún queda espacio suficiente en  */
  /*    el almac. int. para el siguiente registro y bloquear/grabar/ */
  /*    desbloquear el archivo e inicializar el almac.*/
  /*    intermedio si no es así                       */

  do {
     EXEC SQL FETCH c1 INTO :first_name, :last_name, :job_code;
      buffer_len += sprintf( write_ptr, "%s %s %s\n",
                             first_name, last_name, job_code );
      buffer_len = strlen( file_buf ) ;
     /* Grabar el contenido del almac. int. en archivo */
     /* si el almac. int. alcanza el límite            */
     if ( buffer_len >= ( BUF_SIZE - MAX_RECORD_SIZE ) ) {
      /*  obtener bloqueo exclusivo de grabación */
      lock_rc = fcntl( iFileHandle, lock_command, &lock );
 if ( lock_rc != 0 ) goto file_lock_err;
 /*  posicionar al final del archivo */
 lock_rc = lseek( iFileHandle, 0, SEEK_END );
if ( lock_rc < 0 ) goto file_seek_err;
/* grabar el almacenamiento intermedio */
lock_rc = write( iFileHandle,
                             ( void * ) file_buf, buffer_len );
if ( lock_rc < 0 ) goto file_write_err;
  /* liberar el bloqueo */
 lock_rc = fcntl( iFileHandle, lock_command, &unlock );
 if ( lock_rc != 0 ) goto file_unlock_err;
 file_buf[0] = '\0' ;  /* restablecer almacenamiento intermedio */
  buffer_len = 0 ;  /* restablecer longitud del almacenamiento intermedio */
  write_ptr = file_buf ;  /* restablecer puntero de grabación */
     }
                 else
        write_ptr = file_buf + buffer_len ;  /* siguiente posición de grabación */
     }
  } while (1) ;
```

```
cls:
    /* Grabar la última porción de datos en el archivo */
    if (buffer_len > 0) {
       lock_rc = fcntl(iFileHandle, lock_command, &lock);
       if (lock_rc != 0) goto file_lock_err;
      lock_rc = lseek(iFileHandle, 0, SEEK_END);
       if (lock_rc < 0) goto file_seek_err;
       lock_rc = write(iFileHandle, (void *)file_buf, buffer_len);
       if (lock_rc < 0) goto file_write_err;
       lock_rc = fcntl(iFileHandle, lock_command, &unlock);
      if (lock_rc != 0) goto file_unlock_err;
    }
    free(file_buf);
 close(iFileHandle);
    EXEC SQL CLOSE c1;
    exit (0);
 ext:
    if ( SQLCODE != 0 )
       printf( "Error:  SQLCODE = %ld.\n", SQLCODE );
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    EXEC SQL CONNECT RESET;
    if ( SQLCODE != 0 ) {
       printf( "CONNECT RESET Error:  SQLCODE = %ld\n", SQLCODE );
       exit(4);
    }
    exit (5);
 file_lock_err:
    printf("Error: file lock error = %ld.\n",lock_rc);
      /* desbloqueo incondicional del archivo */
   fcntl(iFileHandle, lock_command, &unlock);
    exit(6);
file_seek_err:
    printf("Error: file seek error = %ld.\n",lock_rc);
      /* desbloqueo incondicional del archivo */
   fcntl(iFileHandle, lock_command, &unlock);
   exit(7);
file_write_err:
    printf("Error: file write error = %ld.\n",lock_rc);
      /* desbloqueo incondicional del archivo */
   fcntl(iFileHandle, lock_command, &unlock);
    exit(8);
file_unlock_err:
    printf("Error: file unlock error = %ld.\n",lock_rc);
      /* desbloqueo incondicional del archivo */
   fcntl(iFileHandle, lock_command, &unlock);
    exit(9);
}
```

Este método no sólo es aplicable a la selección de una única tabla, sino que también sirve para consultas más complejas. No obstante, si la consulte requiere operaciones no distribuidas (es decir, Explain muestra más de una subsección junto a la subsección Coordinator), puede tener como consecuencia que se produzca un número excesivo de procesos en algunas particiones, en

caso de que la consulta se ejecute en paralelo en todas las particiones. En esta situación, puede almacenar el resultado de la consulta en una tabla temporal TEMP, en tantas particiones como sea necesario, y luego realizar la extracción final en paralelo desde TEMP.

Si desea extraer todos los empleados, pero únicamente para clasificaciones de trabajo seleccionadas, puede definir la tabla TEMP con los nombres de columna FIRSTNME, LASTNAME y JOB, tal como sigue:

```
   INSERT INTO TEMP
SELECT FIRSTNME, LASTNAME, JOB FROM EMPLOYEE WHERE WORKDEPT IS NOT NULL
       AND EMPNO NOT IN (SELECT EMPNO FROM EMP_ACT WHERE
                         EMPNO<200)
```

A continuación, realizará la extracción paralela sobre TEMP.

Cuando defina la tabla TEMP, tenga en cuenta lo siguiente:

- Si la consulta especifica una agregación GROUP BY, debe definir la clave de particionamiento de TEMP como subconjunto de las columnas GROUP BY.
- La clave de particionamiento de la tabla TEMP debe tener la cardinalidad (es decir, el número de valores diferenciados en el conjunto de respuestas) suficiente para asegurarse de que la tabla se distribuye de forma igualitaria en las particiones en que está definida.
- Cree la tabla TEMP con el atributo NOT LOGGED INITIALLY, luego confirme (COMMIT) la unidad de trabajo que ha creado la tabla para liberar los posibles bloqueos de catálogo adquiridos.
- Cuando utilice la tabla TEMP, debe emitir las sentencias siguientes en una sola unidad de trabajo:
  1. ALTER TABLE TEMP ACTIVATE NOT LOGGED INITIALLY WITH EMPTY TABLE (para vaciar la tabla TEMP y desactivar el registro)
  2. INSERT INTO TEMP SELECT FIRSTNAME...
  3. COMMIT

  Esta técnica le permite insertar un gran conjunto de respuestas en una tabla sin realizar un registro y sin que se produzca ninguna contención de catálogos. Sin embargo, si una tabla tiene activado el atributo NOT LOGGED INITIALLY, se produce una actividad no registrada y una de las siguientes situaciones:

  – Una sentencia falla, causando una retrotracción
  – Se ejecuta una operación ROLLBACK TO SAVEPOINT

  se retrotrae la unidad de trabajo entera (SQL1476N), con lo que la tabla TEMP queda inutilizable. Si así sucede, deberá eliminar y volver a crear la tabla TEMP. Por este motivo, *no* debe utilizar esta técnica para añadir datos a una tabla que no pueda volver a crear fácilmente.

Si necesita que el conjunto final de respuestas (que está formado por el conjunto de respuestas parciales fusionadas de todas las particiones) esté clasificado, puede:

- Especificar la cláusula SORT BY en la sentencia SELECT final
- Realizar una extracción en un archivo separado en cada partición
- Fusionar los archivos separados en un conjunto de salida utilizando, por ejemplo, el mandato `sort -m` de AIX.

Creación de un entorno simulado de bases de datos particionadas

Puede crear un entorno de prueba para sus aplicaciones de entorno particionadas sin configurar un entorno de bases de datos particionadas.

Procedimiento:

Para crear un entorno simulado de bases de datos particionadas:

1. Cree un modelo de diseño de la base de datos con DB2 Enterprise Server Edition.
2. Cree tablas de ejemplo con la cláusula PARTITIONING KEY, que utilizará para distribuir los datos entre las particiones del entorno de producción.
3. Cree y ejecute sus aplicaciones para la base de datos de prueba.

DB2 Enterprise Server Edition impone las restricciones de clave de particionamiento que se aplican en un entorno de bases de datos particionadas y proporciona un útil entorno de prueba para las aplicaciones.

Resolución de problemas

Las secciones siguientes describen cómo solucionar problemas de aplicaciones en un entorno de bases de datos particionadas.

Consideraciones sobre el manejo de errores en entornos de bases de datos particionadas

En un entorno de bases de datos particionadas, DB2 divide las sentencias de SQL en subsecciones, cada una de las cuales se procesa en la partición que contiene los datos relevantes. Como resultado, se puede producir un error en una partición que no tenga acceso a la aplicación. Esta condición no se produce en un entorno de bases de datos no particionadas.

Debe tener en cuenta lo siguiente:

- Errores no graves de no CURSOR (EXECUTE)
- Errores no graves de CURSOR
- Errores graves

- Varias estructuras SQLCA fusionadas
- Cómo identificar la partición que ha devuelto el error

Si una aplicación finaliza de forma anormal debido a un error grave, es posible que se dejen transacciones dudosas en la base de datos. (Una transacción dudosa pertenece a transacciones globales cuando una fase finaliza satisfactoriamente, pero el sistema falla antes de poder completar la siguiente fase, dejando la base de datos en un estado incoherente.)

Conceptos relacionados:

- "Errores graves en entornos de bases de datos particionadas" en la página 494
- "Varias estructuras SQLCA fusionadas" en la página 495
- "Partición que devuelve el error" en la página 496

Tareas relacionadas:

- "Manually resolving indoubt transactions" en el manual *Administration Guide: Planning*

Errores graves en entornos de bases de datos particionadas

Si se produce un error grave en un entorno de bases de datos particionadas, se producirá una de las siguientes situaciones:

- El gestor de bases de datos de la partición en la que se produce el error se cierra.

  Las unidades de trabajo activas no se retrotraen.

  En esta situación, debe recuperar la partición y las bases de datos que estaban activas en la partición cuando se produjo el cierre.
- Se impone la salida de todos los agentes de la base de datos en la partición en la que se ha producido el error.

  Todas las unidades de trabajo de dicha base de datos se retrotraen.

  En esta situación, la partición de base de datos en la que se ha producido el error se marca como incoherente. Cualquier intento de acceder a la misma da como resultado la devolución de SQLCODE -1034 (SQLSTATE 58031) o SQLCODE -1015 (SQLSTATE 55025). Antes de que el usuario o cualquier otra aplicación de otra partición pueda acceder a esta partición de base de datos, debe ejecutar el mandato RESTART DATABASE contra la base de datos.

El error grave SQLCODE -30081 (SQLSTATE 08001) se puede producir por varios motivos. Si recibe este mensaje, compruebe la SQLCA, la cual indicará qué partición ha fallado. Luego compruebe el archivo de registro de notificaciones del administrador para ver detalles.

Conceptos relacionados:

- “Partición que devuelve el error” en la página 496

Consulta relacionada:

- “Mandato RESTART DATABASE” en el manual *Consulta de mandatos*

Varias estructuras SQLCA fusionadas

Distintos agentes de distintas partición de base de datos pueden ejecutar una sentencia de SQL, y cada agente puede devolver una SQLCA distinta para diferentes errores o avisos. El agente coordinador también tiene su propia SQLCA. Además, la SQLCA también tiene campos que indican números globales (como los campos *sqlerrd* que indican números de filas). Para ofrecer una vista coherente para las aplicaciones, todos los valores de SQLCA se fusionan en una estructura.

La notificación de errores se realiza del siguiente modo:

- Las condiciones de errores graves siempre se notifican. En cuanto se notifica un error grave, no se realizan adiciones, además del error grave, a la SQLCA.
- Si no se produce ningún error grave, un error de punto muerto tiene precedencia sobre los demás errores.
- Para los demás errores, se devuelve a la aplicación la SQLCA correspondiente al primer SQLCODE negativo.
- Si no se detecta ningún SQLCODE negativo, se devuelve a la aplicación la SQLCA correspondiente al primer aviso (es decir, SQLCODE positivo). Se produce una excepción a esta norma si se emite una operación de manipulación de datos en una tabla que está vacía en una partición pero que tiene datos en otras particiones. El SQLCODE +100 sólo se devuelve a la aplicación si los agentes de todas las particiones devuelven SQL0100W porque la tabla está vacía en todas las particiones o porque no hay filas que satisfagan la cláusula WHERE de una sentencia UPDATE.
- Para todos los errores y avisos, el campo *sqlwarn* contiene los distintivos de aviso recibidos de todos los agentes.
- Los valores de los campos *sqlerrd* que indican números de filas son sumas de todos los agentes.

Una aplicación puede recibir un error o aviso siguiente después de que se corrija el problema que ocasionó el primer error o aviso. Los errores se notifican a la SQLCA para asegurar que el primer error detectado tiene prioridad sobre otros. Esto asegura que un error ocasionado por un error anterior no puede sobregrabar el error original. Los errores graves y los errores de punto muerto tienen una prioridad más alta porque necesitan una acción inmediata por parte del agente coordinador.

Consulta relacionada:

- "SQLCA" en el manual *Administrative API Reference*

Partición que devuelve el error

Si una partición devuelve un error o aviso, su número está en el campo SQLERRD(6) de la SQLCA. El número de este campo coincide con el especificado para la partición en el archivo `db2nodes.cfg`.

Si una llamada a una sentencia de SQL o API resulta satisfactoria, el número de partición de este campo no es significativo.

Consulta relacionada:

- "SQLCA" en el manual *Administrative API Reference*

Aplicaciones en bucle o suspendidas

Es posible que, después de iniciar una consulta o aplicación, sospeche que está suspendida (no muestra ninguna actividad) o que ha entrado en un bucle (muestra actividad, pero no se devuelve ningún resultado a la aplicación). Asegúrese de que ha activado los tiempos de espera de bloqueo. Sin embargo, en algunas situaciones no se devuelve ningún error. En estas situaciones, pueden resultarle de ayuda las herramientas de diagnóstico que se proporcionan con y la instantánea del supervisor del sistema de bases de datos.

Una de las funciones del supervisor del sistema de bases de datos que resulta útil para depurar aplicaciones consiste en visualizar el estado de todos los agentes activos. Para sacar el máximo provecho de una instantánea, asegúrese de que la recopilación de sentencias se realiza antes de ejecutar la aplicación (preferiblemente inmediatamente después de ejecutar DB2START) del siguiente modo:

```
db2_all "db2 UPDATE MONITOR SWITCHES USING STATEMENT ON"
```

Si sospecha que la aplicación o consulta está atascada o ha entrado en un bucle, emita el siguiente mandato:

```
db2_all "db2 GET SNAPSHOT FOR AGENTS ON base de datos
```

Conceptos relacionados:

- "Database system monitor" en el manual *System Monitor Guide and Reference*
- "The database system-monitor information" en el manual *Administration Guide: Performance*

Consulta relacionada:

- "Mandato GET SNAPSHOT" en el manual *Consulta de mandatos*

- “Mandato UPDATE MONITOR SWITCHES” en el manual *Consulta de mandatos*
- “db2trc - Mandato Rastrear” en el manual *Consulta de mandatos*
- “db2support - Mandato Herramienta de análisis de problemas y recolección del entorno” en el manual *Consulta de mandatos*

Capítulo 18. Técnicas comunes de aplicación de DB2

Columnas generadas

En lugar de utilizar engorrosos activadores de inserción y actualización, DB2 le permite incluir columnas generadas en sus tablas mediante la cláusula GENERATED ALWAYS AS. Una columna generada es una columna que deriva de los valores de cada fila procedente de una expresión, en lugar de hacerlo de una operación de inserción o actualización. Aunque que la combinación de un activador de actualización y un activador de inserción puede conseguir un efecto parecido, la utilización de una columna generada garantiza que el valor derivado es coherente con la expresión.

Para crear una columna generada en una tabla, utilice la cláusula GENERATED ALWAYS AS para la columna e incluya la expresión de la cual se derivará el valor correspondiente a la columna. Puede incluir la cláusula GENERATED ALWAYS AS en sentencias ALTER TABLE y CREATE TABLE. El siguiente ejemplo crea una tabla con dos columnas normales, “c1” y “c2”, y dos columnas generadas, “c3” y “c4”, que se derivan de las columnas normales de la tabla.

```
CREATE TABLE T1(c1 INT, c2 DOUBLE,
                c3 DOUBLE GENERATED ALWAYS AS (c1 + c2),
                c4 GENERATED ALWAYS AS
                  (CASE
                     WHEN c1 > c2 THEN 1
                     ELSE NULL
                   END)
               );
```

Tareas relacionadas:

- "Defining a generated column on a new table" en el manual *Administration Guide: Implementation*
- "Defining a generated column on an existing table" en el manual *Administration Guide: Implementation*

Consulta relacionada:

- "ALTER TABLE sentencia" en el manual *Consulta de SQL, Volumen 2*
- "CREATE TABLE sentencia" en el manual *Consulta de SQL, Volumen 2*

Ejemplos relacionados:

- "TbGenCol.java -- How to use generated columns (JDBC)"

Columnas de identidad

Las columnas de identidad proporcionan a los programadores de aplicaciones de DB2 una forma sencilla de generar automáticamente un valor numérico de columna para cada fila de una tabla. Puede hacer que este valor se genere como valor exclusivo y luego definir la columna de identidad como la clave principal para la tabla. Para crear una columna de identidad, incluya la cláusula IDENTITY en la sentencia CREATE TABLE o ALTER TABLE.

Utilice columnas de identidad en las aplicaciones para evitar problemas de concurrencia y de rendimiento que se pueden producir cuando una aplicación genera su propio contador exclusivo fuera de la base de datos. Cuando no utilice columnas de identidad para generar automáticamente claves principales exclusivas, un diseño común consiste en almacenar un contador en una tabla con una sola fila. Luego cada transacción bloquea esta tabla, aumenta el número y luego confirma la transacción para desbloquear el contador. Desgraciadamente, este diseño sólo permite que una sola transacción aumente el contador cada vez.

Por el contrario, si utiliza una columna de identidad para generar automáticamente claves principales, la aplicación puede conseguir niveles de concurrencia muy superiores. Con columnas de identidad, DB2 mantiene el contador de forma que las transacciones no lo tienen que bloquear. Las aplicaciones que utilizan columnas de identidad pueden funcionar mejor porque una transacción no confirmada que ha aumentado el contador no evita que otras transacciones siguientes puedan también aumentar el contador.

El contador correspondiente a la columna de identidad aumenta o desciende independientemente de la transacción. Si una determinada transacción aumenta un contador de identidad dos veces, dicha transacción puede

observar un salto en los dos números que se generan porque es posible que otras transacciones estén incrementando simultáneamente el mismo contador de identidad.

Puede que parezca que una columna de identidad ha generado saltos en el contador, como resultado de una transacción que se ha retrotraído o porque la base de datos ha colocado en antememoria un rango de valores que se han desactivado (de forma normal o anómala) antes de que se asignaran todos los valores colocados en antememoria.

Para recuperar el valor generado después de insertar una fila nueva en una tabla con una columna de identidad, utilice la función `identity_val_local()`.

La cláusula IDENTITY está disponible para las sentencias CREATE TABLE y ALTER TABLE.

Conceptos relacionados:

- "Identity columns" en el manual *Administration Guide: Planning*

Tareas relacionadas:

- "Defining an identity column on a new table" en el manual *Administration Guide: Implementation*
- "Modifying an identity column definition" en el manual *Administration Guide: Implementation*
- "Altering an identity column" en el manual *Administration Guide: Implementation*

Consulta relacionada:

- "ALTER TABLE sentencia" en el manual *Consulta de SQL, Volumen 2*
- "CREATE TABLE sentencia" en el manual *Consulta de SQL, Volumen 2*

Ejemplos relacionados:

- "tbident.sqc -- How to use identity columns (C)"
- "TbIdent.java -- How to use Identity Columns (JDBC)"
- "TbIdent.sqlj -- How to use Identity Columns (SQLj)"

Valores secuenciales y objetos de secuencia

Las secciones siguientes describen consideraciones sobre valores secuenciales y objetos de secuencia.

Generación de valores secuenciales

La generación de valores secuenciales es un problema común del desarrollo de aplicaciones de base de datos. La mejor solución a este problema es la utilización de objetos de secuencia y expresiones de secuencia en SQL. Cada *objeto de secuencia* es un objeto de base de datos con nombre exclusivo al que sólo pueden acceder las expresiones de secuencia. Hay dos *expresiones de secuencia*: la expresión PREVVAL y la expresión NEXTVAL. La expresión PREVVAL devuelve el último valor generado en el proceso de aplicación correspondiente al objeto de secuencia especificado. Cualquier expresión NEXTVAL que aparezca en la misma sentencia que la expresión PREVAL no tiene ningún efecto sobre el valor que genera la expresión PREVAL en dicha sentencia. La expresión de secuencia NEXTVAL incrementa el valor del objeto de secuencia y devuelve el nuevo valor del objeto de secuencia.

Para crear un objeto de secuencia, emita la sentencia CREATE SEQUENCE. Por ejemplo, para crear un objeto de secuencia llamado id_values utilizando los atributos por omisión, emita la sentencia siguiente:

```
CREATE SEQUENCE id_values
```

Para generar el primer valor de la sesión de la aplicación correspondiente al objeto de secuencia, emita una sentencia VALUES que utilice la expresión NEXTVAL:

```
VALUES NEXTVAL FOR id_values

1
-----------
          1

  1 registro(s) seleccionado(s).
```

Para visualizar el valor actual del objeto de secuencia, emita una sentencia VALUES que utilice la expresión PREVVAL:

```
VALUES PREVVAL FOR id_values

1
-----------
          1

  1 registro(s) seleccionado(s).
```

Se puede recuperar el valor actual del objeto de secuencia repetidamente y el valor que el objeto de secuencia devuelve no cambia hasta que se emite una expresión NEXTVAL. En el ejemplo siguiente, la expresión PREVVAL devuelve un valor de 1, hasta que la expresión NEXTVAL de la conexión actual aumenta el valor del objeto de secuencia:

```
VALUES PREVVAL FOR id_values

1
-----------
          1

            1 registro(s) seleccionado(s).

VALUES PREVVAL FOR id_values

1
-----------
          1

            1 registro(s) seleccionado(s).

VALUES NEXTVAL FOR id_values

1
-----------
          2

            1 registro(s) seleccionado(s).

VALUES PREVVAL FOR id_values

1
-----------
          2

            1 registro(s) seleccionado(s).
```

Para actualizar el valor de una columna con el siguiente valor del objeto de secuencia, incluya la expresión NEXTVAL en la sentencia UPDATE, del modo siguiente:

```
UPDATE personal
  SET id = NEXTVAL FOR id_values
  WHERE id = 350
```

Para insertar una nueva fila en una tabla utilizando el siguiente valor del objeto de secuencia, incluya la expresión NEXTVAL en la sentencia INSERT, del modo siguiente:

```
INSERT INTO personal (id, nombre, dept, cargo)
  VALUES (NEXTVAL FOR id_values, 'Kandil', 51, 'Director')
```

Consulta relacionada:

- "CREATE SEQUENCE sentencia" en el manual *Consulta de SQL, Volumen 2*

Ejemplos relacionados:

- "DbSeq.java -- How to create, alter and drop a sequence in a database (JDBC)"

Gestión del comportamiento de secuencias

Puede adaptar el comportamiento de objetos de secuencia para satisfacer las necesidades de su aplicación. Se cambian los atributos de un objeto de secuencia cuando se emite la sentencia CREATE SEQUENCE para crear un nuevo objeto de secuencia y cuando se emite una sentencia ALTER SEQUENCE para un objeto de secuencia ya existente. A continuación se encuentran algunos de los atributos de un objeto de secuencia que se pueden especificar:

Tipo de datos
La cláusula AS de la sentencia CREATE SEQUENCE especifica el tipo de datos numérico del objeto de secuencia. El tipo de datos determina los posibles valores mínimo y máximo del objeto de secuencia (los valores mínimo y máximo correspondientes a un tipo de datos se listan en el tema que describe los límites de SQL). No se puede cambiar el tipo de datos de un objeto de secuencia; en su lugar, deberá descartar el objeto de secuencia emitiendo la sentencia DROP SEQUENCE y emitir una sentencia CREATE SEQUENCE con el nuevo tipo de datos.

Valor inicial
La cláusula START WITH de la sentencia CREATE SEQUENCE establece el valor inicial del objeto de secuencia. La cláusula RESTART WITH de la sentencia ALTER SEQUENCE restablece el valor del objeto de secuencia en un valor especificado.

Valor mínimo
La cláusula MINVALUE establece el valor mínimo del objeto de secuencia.

Valor máximo
La cláusula MAXVALUE establece el valor máximo del objeto de secuencia.

Valor de incremento
La cláusula INCREMENT BY establece el valor que cada expresión NEXTVAL añade al valor actual del objeto de secuencia. Para disminuir el valor del objeto de secuencia, especifique un valor negativo.

Secuencia cíclica
La cláusula CYCLE hace que el valor de un objeto de secuencia que alcanza su valor máximo o mínimo genere su valor mínimo o máximo respectivo en la siguiente expresión NEXTVAL.

Por ejemplo, para crear un objeto de secuencia llamado id_values que empiece con un valor mínimo de 0, tenga un valor máximo de 1000, incremente en 2

con cada expresión NEXTVAL y vuelva a su valor mínimo cuando se alcance el valor máximo, emita la sentencia siguiente:

```
CREATE SEQUENCE id_values
  START WITH 0
  INCREMENT BY 2
  MAXVALUE 1000
  CYCLE
```

Consulta relacionada:

- "Límites de SQL" en el manual *Consulta de SQL, Volumen 1*
- "ALTER SEQUENCE sentencia" en el manual *Consulta de SQL, Volumen 2*
- "CREATE SEQUENCE sentencia" en el manual *Consulta de SQL, Volumen 2*

Rendimiento de la aplicación y objetos de secuencia

Al igual que las columnas de identidad, la utilización de objetos de secuencia para generar valores normalmente mejora el rendimiento de las aplicaciones en comparación con los acercamientos alternativos. La alternativa a los objetos de secuencia consiste en crear una tabla de una sola columna que almacene el valor actual e incremente este valor con un activador o bajo el control de la aplicación. En un entorno distribuido donde las aplicaciones acceden de forma simultánea a la tabla de una sola columna, el bloqueo necesario para forzar el acceso en serie a la tabla puede afectar gravemente al rendimiento.

Los objetos de secuencia evitan los problemas de bloqueo que están asociados con el enfoque de tablas de una única columna y pueden almacenar en antememoria valores de secuencia en memoria para mejorar el tiempo de respuesta de DB2. Para maximizar el rendimiento de aplicaciones que utilizan objetos de secuencia, asegúrese de que el objeto de secuencia almacene en antememoria una cantidad apropiada de valores de secuencia. La cláusula CACHE de las sentencias CREATE SEQUENCE y ALTER SEQUENCE especifica el número máximo de valores de secuencia que DB2 genera y almacena en memoria.

Si el objeto de secuencia debe generar valores en orden, sin incorporar espacios en ese orden debido a una anomalía del sistema o desactivación de la base de datos, utilice las cláusulas ORDER y NO CACHE en la sentencia CREATE SEQUENCE. La cláusula NO CACHE garantiza que no aparezcan espacios en los valores generados a expensas de parte del rendimiento de la aplicación, puesto que fuerza al objeto de secuencia a grabar en la anotación cronológica de la base de datos cada vez que genera un nuevo valor. Observe que es posible que sigan apareciendo saltos debido a transacciones que se retrotraen y no utilizan realmente el valor de secuencia que han solicitado.

Comparación entre objetos de secuencia y columnas de identidad

Aunque los objetos de secuencia y las columnas de identidad parecen tener finalidades parecidas para aplicaciones de DB2, existe una diferencia importante. Una columna de identidad genera automáticamente valores para una columna en una sola tabla. Un objeto de secuencia genera valores secuenciales bajo petición que se pueden utilizar en cualquier sentencia de SQL.

Tablas temporales declaradas y rendimiento de la aplicación

Una *tabla temporal declarada* es una tabla temporal a la que sólo pueden acceder las sentencias de SQL que emite la aplicación que ha creado la tabla temporal. Una tabla temporal declarada no dura más que la conexión de la aplicación con la base de datos.

Utilice tablas temporales declaradas para mejorar potencialmente el rendimiento de sus aplicaciones. Cuando crea una tabla temporal declarada, DB2 no inserta una entrada en las tablas del catálogo del sistema, y por lo tanto el servidor no experimenta problemas relacionados con la contención por el catálogo. En comparación con las tablas normales, DB2 no bloquea las tablas temporales declaradas ni sus filas y, si especifica el parámetro NOT LOGGED al crearlas, no registra las tablas temporales declaradas ni su contenido. Si la aplicación actual crea tablas para procesar grandes cantidades de datos y elimina dichas tablas cuando la aplicación ha terminado de manipular dichos datos, considere la posibilidad de utilizar tablas temporales declaradas en lugar de tablas normales.

Si desarrolla aplicaciones escritas para usuarios simultáneos, las aplicaciones se pueden beneficiar de las tablas temporales declaradas. A diferencia de las tablas normales, las tablas temporales declaradas no están sujetas a colisión de nombres. Para cada instancia de la aplicación, DB2 puede crear una tabla temporal declarada con un nombre idéntico. Por ejemplo, para escribir una aplicación para usuarios simultáneos que utilice tablas normales para procesar grandes cantidades de datos temporales, debe asegurarse de que cada instancia de la aplicación utilice un nombre exclusivo para la tabla normal que contiene los datos temporales. Normalmente, crearía otra tabla que efectúe un seguimiento de los nombres de las tablas que se están utilizando en cada momento. Con tablas temporales declaradas, simplemente especifique un nombre de tabla temporal declarada para sus datos temporales. DB2 garantiza que cada instancia de la aplicación utiliza una tabla exclusiva.

Para utilizar una tabla temporal declarada, siga los pasos siguientes:

Paso 1. Asegúrese de que existe un USER TEMPORARY TABLESPACE. Si no existe ningún USER TEMPORARY TABLESPACE, emita una sentencia CREATE USER TEMPORARY TABLESPACE.

Paso 2. Emita una sentencia DECLARE GLOBAL TEMPORARY TABLE en la aplicación.

El esquema correspondiente a tablas temporales declaradas siempre es SESSION. Para utilizar la tabla temporal declarada en las sentencias de SQL, debe hacer referencia a la tabla mediante el calificador de esquema SESSION de forma explícita o utilizando un esquema DEFAULT de SESSION para calificar las referencias no calificadas. En el siguiente ejemplo, el nombre de tabla siempre está calificado mediante el nombre de esquema SESSION cuando crea una tabla temporal declarada denominada TT1 con la siguiente sentencia:

```
DECLARE GLOBAL TEMPORARY TABLE TT1
```

Para seleccionar el contenido de la columna *column1* de la tabla temporal declarada creada en el ejemplo anterior, utilice la sentencia siguiente:

```
SELECT column1 FROM SESSION.TT1;
```

Observe que DB2 también le permite crear tablas permanentes con el esquema SESSION. Si crea una tabla permanente con el nombre calificado SESSION.TT3, luego puede crear una tabla temporal declarada con el nombre calificado SESSION.TT3. En esta situación, DB2 siempre resuelve las referencias a las tablas permanente y temporal declarada con nombres calificados idénticos a la tabla temporal declarada. Para evitar confundir tablas permanentes con tablas temporales declaradas, no debe crear tablas permanentes con el esquema SESSION.

Si crea una aplicación que incluye una referencia de SQL estático a una tabla, vista o alias calificado con el esquema SESSION, el precompilador de DB2 no compila dicha sentencia en el momento de la vinculación y marca la sentencia para indicar que “necesita compilación”. En el tiempo de ejecución, DB2 compila la sentencia. Este comportamiento se denomina *vinculación incremental*. DB2 realiza automáticamente una vinculación incremental para referencias de SQL estático a tablas, vistas y alias calificados con el esquema SESSION. No hace falta que especifique la opción VALIDATE RUN en el mandato BIND o PRECOMPILE para permitir una vinculación incremental para dichas sentencias.

Si emite una sentencia ROLLBACK para una transacción que incluye una sentencia DECLARE GLOBAL TEMPORARY TABLE, DB2 elimina la tabla temporal declarada. Si emite una sentencia DROP TABLE para una tabla temporal declarada, al emitir una sentencia ROLLBACK para dicha

transacción sólo se restaura una tabla temporal declarada vacía. Una sentencia ROLLBACK de una sentencia DROP TABLE no restaura las filas que existían en la tabla temporal declarada.

El comportamiento por omisión de una tabla temporal declarada consiste en suprimir todas las filas de la tabla cuando el usuario confirma una transacción. Sin embargo, si quedan uno o más cursores WITH HOLD abiertos en la tabla temporal declarada, DB2 no suprime las filas de la tabla cuando el usuario confirma una transacción. Para evitar suprimir todas las filas cuando confirma una transacción, cree la tabla temporal utilizando la cláusula ON COMMIT PRESERVE ROWS en la sentencia DECLARE GLOBAL TEMPORARY TABLE.

Si modifica el contenido de una tabla temporal declarada mediante una sentencia INSERT, UPDATE o DELETE dentro de una transacción y retrotrae dicha transacción, DB2 suprime todas las filas de la tabla temporal declarada. Si intenta modificar el contenido de una tabla temporal declarada mediante una sentencia INSERT, UPDATE o DELETE y la sentencia falla, DB2 se comporta del siguiente modo:

- Si la tabla se creó sin el parámetro NOT LOGGED (es decir, la tabla se registra), sólo se retrotraen los cambios realizados por la sentencia INSERT, UPDATE o DELETE.
- Si la tabla se creó con el parámetro NOT LOGGED, DB2 suprime todas las filas de la tabla temporal declarada.

Cuando se encuentra una anomalía en un entorno de bases de datos particionadas, todas las tablas temporales declaradas existentes en la partición de base de datos que ha fallado pasan a estar inutilizable. Cualquier acceso siguiente a estas tablas temporales declaradas inutilizables devuelve un error (SQL1477N). Cuando la aplicación encuentra una tabla temporal declarada inutilizable, la aplicación puede eliminar la tabla o volverla a crear especificando la cláusula WITH REPLACE en la sentencia DECLARE GLOBAL TEMPORARY TABLE.

Las tablas temporales declaradas están sujetas a varias restricciones. Por ejemplo, no puede definir alias ni vistas para tablas temporales declaradas. No puede utilizar IMPORT y LOAD para llenar tablas temporales declaradas. Puede, aunque con restricciones, crear índices para tablas temporales declaradas. Además, puede ejecutar RUNSTATS contra una tabla temporal declarada para actualizar estadísticas correspondientes a la tabla temporal declarada y sus índices.

Consulta relacionada:

- "DECLARE GLOBAL TEMPORARY TABLE sentencia" en el manual *Consulta de SQL, Volumen 2*

Ejemplos relacionados:

- "tbtemp.sqc -- How to use a declared temporary table (C)"
- "TbTemp.java -- How to use Declared Temporary Table (JDBC)"

Puntos de rescate y transacciones

Las secciones siguientes describen los puntos de rescate y cómo utilizarlos para gestionar transacciones.

Gestión de transacciones con puntos de rescate

Los puntos de rescate de aplicaciones proporcionan control sobre el trabajo realizado por un subconjunto de sentencias de SQL en una transacción o unidad de trabajo. Dentro de la aplicación puede definir un punto de rescate y luego liberarlo o retrotraer el trabajo realizado desde que definió el punto de rescate. Puede utilizar tantos puntos de rescate como necesite dentro de una sola transacción; sin embargo, no puede anidar puntos de rescate. El siguiente ejemplo muestra el uso de dos puntos de rescate dentro de una sola transacción para controlar el comportamiento de una aplicación:

Ejemplo de un pedido que utiliza puntos de rescate de la aplicación:

```
INSERT INTO order ...
INSERT INTO order_item ... lamp

-- definir el primer punto de rescate de la transacción
SAVEPOINT before_radio ON ROLLBACK RETAIN CURSORS
  INSERT INTO order_item ... Radio
  INSERT INTO order_item ... Power Cord
  -- Pseudo-SQL:
  IF SQLSTATE = "No Power Cord"
    ROLLBACK TO SAVEPOINT before_radio
RELEASE SAVEPOINT before_radio

-- definir el segundo punto de rescate de la transacción
SAVEPOINT before_checkout ON ROLLBACK RETAIN CURSORS
  INSERT INTO order ... Approval
  -- Pseudo-SQL:
  IF SQLSTATE = "No approval"
    ROLLBACK TO SAVEPOINT before_checkout

-- confirmar la transacción, lo cual libera el punto de rescate
COMMIT
```

En el ejemplo anterior, el primer punto de rescate hace cumplir una dependencia entre dos objetos de datos en los que la dependencia no es intrínseca de los propios objetos. No utilizaría integridad referencial para describir la relación anterior entre radios y cables de alimentación (power cords) puesto que un objeto puede existir sin el otro. Sin embargo, no desea distribuir al cliente la radio sin un cable de alimentación. Tampoco desea

cancelar el pedido de la lámpara (lamp) retrotrayendo la transacción entera porque no hay cables de alimentación para la radio. Los puntos de rescate de la aplicación proporcionan el control granular que necesita para completar este pedido.

Cuando emite una sentencia ROLLBACK TO SAVEPOINT, el punto de rescate correspondiente no se libera automáticamente. Las siguientes sentencias de SQL se asocian con este punto de rescate hasta que este se libera de forma explícita con una sentencia RELEASE SAVEPOINT o de forma implícita finalizando la transacción o unidad de trabajo. Esto significa que puede emitir varias sentencias ROLLBACK TO SAVEPOINT para un solo punto de rescate.

Los puntos de rescate le ofrecen un mejor rendimiento y un diseño de aplicación más limpio que la utilización de varias sentencias COMMIT y ROLLBACK. Cuando emite una sentencia COMMIT, DB2 debe realizar un trabajo adicional para confirmar la transacción actual e iniciar una nueva transacción. Los puntos de rescate le permiten dividir una transacción en unidades de menor tamaño o pasos sin el proceso general añadido de varias sentencias COMMIT. El siguiente ejemplo muestra el deterioro del rendimiento que se produce si se utilizan varias transacciones en lugar de puntos de rescate:

Ejemplo de un pedido que utiliza varias transacciones::

```
INSERT INTO order ...
INSERT INTO order_item ... lamp
-- confirmar transacción actual, iniciar transacción nueva
COMMIT

INSERT INTO order_item ... Radio
INSERT INTO order_item ... Power Cord
-- Pseudo-SQL:
IF SQLSTATE = "No Power Cord"
  -- retrotraer transacción actual, iniciar transacción nueva
  ROLLBACK
ELSE
  -- confirmar transacción actual, iniciar transacción nueva
  COMMIT

INSERT INTO order ... Approval
-- Pseudo-SQL:
IF SQLSTATE = "No approval"
  -- retrotraer transacción actual, iniciar transacción nueva
  ROLLBACK
ELSE
  -- confirmar transacción actual, iniciar transacción nueva
COMMIT
```

Otro inconveniente de utilizar varios puntos de confirmación es que se puede confirmar un objeto, con lo cual estará visible para otras aplicaciones antes de

que haya finalizado por completo. En el segundo ejemplo, el pedido está disponible para otro usuario antes de que se hayan añadido todos los elementos y, lo que es peor, antes de que se haya aprobado. El uso de puntos de rescate de la aplicación evita esta exposición a 'datos sucios' mientras se ofrece un control granular sobre una operación.

Ejemplos relacionados:

- "tbsavept.sqc -- How to use external savepoints (C)"

Comparación entre puntos de rescate de la aplicación y bloques de SQL compuesto

Los puntos de rescate ofrecen las siguientes ventajas sobre los bloques de SQL compuesto:

- Control mejorado de transacciones
- Menor contención de bloqueo
- Integración mejorada con la lógica de la aplicación

Los bloques de SQL compuesto pueden ser de tipo ATOMIC o NOT ATOMIC. Si una sentencia dentro de un bloque de SQL compuesto de tipo ATOMIC falla, el bloque de SQL compuesto entero se retrotrae. Si una sentencia dentro de un bloque de SQL compuesto de tipo NOT ATOMIC falla, la aplicación controla la confirmación o retrotracción de la aplicación, incluido el bloque de SQL compuesto entero. En comparación, si una sentencia dentro del ámbito de un punto de rescate falla, la aplicación puede retrotraer todas las sentencias del ámbito del punto de rescate pero confirmar el trabajo realizado por sentencias externas al ámbito del punto de rescate. Esta opción se ilustra en el siguiente ejemplo. Si el trabajo del punto de rescate se retrotrae, el trabajo de las dos sentencias INSERT anteriores al punto de rescate se confirma. Como alternativa, la aplicación puede confirmar el trabajo realizado por todas las sentencias de la transacción, incluidas las sentencias dentro del ámbito del punto de rescate.

Ejemplo de un pedido que utiliza puntos de rescate de la aplicación:

```
INSERT INTO order ...
INSERT INTO order_item ... lamp

-- definir el primer punto de rescate de la transacción
SAVEPOINT before_radio ON ROLLBACK RETAIN CURSORS
    INSERT INTO order_item ... Radio
    INSERT INTO order_item ... Power Cord
    -- Pseudo-SQL:
    IF SQLSTATE = "No Power Cord"
      ROLLBACK TO SAVEPOINT before_radio
RELEASE SAVEPOINT before_radio

-- definir el segundo punto de rescate de la transacción
SAVEPOINT before_checkout ON ROLLBACK RETAIN CURSORS
```

```
  INSERT INTO order ... Approval
    -- Pseudo-SQL:
  IF SQLSTATE = "No approval"
    ROLLBACK TO SAVEPOINT before_checkout

-- confirmar la transacción, lo cual libera el punto de rescate
COMMIT
```

Cuando emite un bloque de SQL compuesto, DB2 adquiere simultáneamente los bloqueos necesarios para el bloque completo de sentencias de SQL compuesto. Cuando define un punto de rescate de la aplicación, DB2 adquiere bloqueos a medida que se emite cada sentencia en el ámbito del punto de rescate. Este comportamiento de bloqueo de puntos de rescate puede llevar a una contención de bloqueo significativamente menor que la que ocasionan los bloques de SQL compuesto, de modo que, a menos que la aplicación necesite que el bloqueo lo realicen las sentencias de SQL compuesto, puede ser aconsejable utilizar puntos de rescate.

Los bloques de SQL compuesto ejecutan un conjunto completo de sentencias como una sola sentencia. Una aplicación no puede utilizar funciones o estructuras de control para añadir sentencias a un bloque de SQL compuesto. En comparación, cuando define un punto de rescate de la aplicación, esta puede emitir sentencias de SQL dentro del ámbito del punto de rescate realizando llamadas a otros métodos o funciones de aplicación, a través de estructuras de control como bucles while o con sentencias de SQL dinámico. Los puntos de rescate de la aplicación le ofrecen libertad para integrar las sentencias de SQL con la lógica de la aplicación de forma intuitiva.

Por ejemplo, en el ejemplo siguiente la aplicación define un punto de rescate y emite dos sentencias INSERT dentro del ámbito del punto de rescate. La aplicación utiliza una sentencia IF que, cuando es cierta, llama a la función add_batteries(). La función add_batteries() emite un sentencia de SQL que en este contexto se incluye dentro del ámbito del punto de rescate. Finalmente, la aplicación retrotrae el trabajo realizado dentro del punto de rescate (incluida la sentencia de SQL que ha emitido la función add_batteries()) o confirma el trabajo realizado en la transacción entera:

Ejemplo de integración de puntos de rescate y sentencias de SQL dentro de la lógica de la aplicación:

```
void add_batteries()
{
  -- el punto de rescate definido en main() controla
  -- el trabajo realizado por la siguiente sentencia
  INSERT INTO order_item ... Batteries
}

void main(int argc, char[] *argv)
{
```

```
  INSERT INTO order ...
  INSERT INTO order_item ... lamp

  -- definir el primer punto de rescate de la transacción
  SAVEPOINT before_radio ON ROLLBACK RETAIN CURSORS
    INSERT INTO order_item ... Radio
    INSERT INTO order_item ... Power Cord

    if (strcmp(Radio..power_source(), "AC/DC"))
    {
      add_batteries();
    }

    -- Pseudo-SQL:
    IF SQLSTATE = "No Power Cord"
      ROLLBACK TO SAVEPOINT before_radio
  COMMIT
}
```

Sentencias de SQL para crear y controlar puntos de rescate

Las siguientes sentencias de SQL le permiten crear y controlar puntos de rescate:

SAVEPOINT

Para definir un punto de rescate, emita una sentencia de SQL SAVEPOINT. Para que el código quede más claro, puede elegir un nombre significativo para el punto de rescate. Por ejemplo:

```
SAVEPOINT savepoint1 ON ROLLBACK RETAIN CURSORS
```

RELEASE SAVEPOINT

Para liberar un punto de rescate, emita una sentencia de SQL RELEASE SAVEPOINT. Si no libera de forma explícita un punto de rescate con una sentencia de SQL RELEASE SAVEPOINT, se libera al final de la transacción. Por ejemplo:

```
RELEASE SAVEPOINT savepoint1
```

ROLLBACK TO SAVEPOINT

Para retrotraer un punto de rescate, emita una sentencia de SQL ROLLBACK TO SAVEPOINT. Por ejemplo:

```
ROLLBACK TO SAVEPOINT
```

Consulta relacionada:

- “ROLLBACK sentencia” en el manual *Consulta de SQL, Volumen 2*
- “RELEASE SAVEPOINT sentencia” en el manual *Consulta de SQL, Volumen 2*
- “SAVEPOINT sentencia” en el manual *Consulta de SQL, Volumen 2*

Ejemplos relacionados:

- “tbsavept.sqc -- How to use external savepoints (C)”

Restricciones sobre el uso de puntos de rescate

DB2 Universal Database impone las siguientes restricciones en el uso de puntos de rescate en aplicaciones:

SQL compuesto atómico
DB2 no le permite utilizar puntos de rescate dentro de SQL compuesto atómico. No puede utilizar SQL compuesto atómico dentro de un punto de rescate.

Puntos de rescate anidados
DB2 no da soporte al uso de un punto de rescate dentro de otro punto de rescate.

Activadores
DB2 no da soporte al uso de puntos de rescate en activadores.

sentencia SET INTEGRITY
Dentro de un punto de rescate, DB2 trata las sentencias SET INTEGRITY como sentencias de DDL.

Conceptos relacionados:

- “Puntos de rescate y Lenguaje de definición de datos (DDL)” en la página 514

Puntos de rescate y Lenguaje de definición de datos (DDL)

DB2 le permite incluir sentencias de DDL dentro de un punto de rescate. Si la aplicación libera satisfactoriamente un punto de rescate que ejecuta sentencias de DDL, la aplicación puede continuar utilizando los objetos SQL creados por DDL. Sin embargo, si la aplicación emite una sentencia ROLLBACK TO SAVEPOINT para un punto de rescate que ejecuta sentencias de DDL, DB2 marca los cursores que dependen de los efectos de dichas sentencias de DDL como no válidos.

En el siguiente ejemplo, la aplicación intenta captar desde de los tres cursores abiertos anteriormente después de emitir una sentencia ROLLBACK TO SAVEPOINT:

```
SAVEPOINT savepoint_name;
   PREPARE s1 FROM 'SELECT FROM t1';
   --emitir sentencia de DDL para t1
      ALTER TABLE t1 ADD COLUMN...
   PREPARE s2 FROM 'SELECT FROM t2';
   --emitir sentencia de DDL para t3
      ALTER TABLE t3 ADD COLUMN...
   PREPARE s3 FROM 'SELECT FROM t3';
   OPEN c1 USING s1;
   OPEN c2 USING s2;
```

```
   OPEN c3 USING s3;
ROLLBACK TO SAVEPOINT
FETCH c1; --no válido (SQLCODE -910)
FETCH c2; --satisfactorio
FETCH c3; --no válido (SQLCODE -910)
```

En la sentencia ROLLBACK TO SAVEPOINT, DB2 marca los cursores “c1” y “c3” como no válidos porque las sentencias de DDL dentro del punto de rescate han manipulado los objetos SQL de los que dependen. Sin embargo, una sentencia FETCH que utiliza el cursor “c2” en el ejemplo es satisfactoria después de la sentencia ROLLBACK TO SAVEPOINT.

Puede emitir una sentencia CLOSE para cerrar los cursores no válidos. Si emite una sentencia FETCH contra un cursor no válido, DB2 devuelve SQLCODE −910. Si emite una sentencia OPEN contra un cursor no válido, DB2 devuelve SQLCODE −502. Si emite una sentencia UPDATE o DELETE WHERE CURRENT OF contra un cursor no válido, DB2 devuelve SQLCODE −910.

Dentro de los puntos de rescate, DB2 trata las tablas con la propiedad NOT LOGGED INITIALLY y las tablas temporales del siguiente modo:

Tablas NOT LOGGED INITIALLY
: Dentro de un punto de rescate, puede crear una tabla con la propiedad NOT LOGGED INITIALLY o modificar una tabla para que tenga la propiedad NOT LOGGED INITIALLY. Sin embargo, para estos puntos de rescate DB2 trata las sentencias ROLLBACK TO SAVEPOINT como sentencias ROLLBACK WORK y retrotrae la transacción entera.

DECLARE TEMPORARY TABLE dentro del punto de rescate
: Si se declara una tabla temporal dentro de un punto de rescate, una sentencia ROLLBACK TO SAVEPOINT elimina la tabla temporal.

DECLARE TEMPORARY TABLE fuera del punto de rescate
: Si se declara una tabla temporal fuera de un punto de rescate, una sentencia ROLLBACK TO SAVEPOINT no elimina la tabla temporal.

Puntos de rescate e inserciones en almacenamiento intermedio

Para mejorar el rendimiento de las aplicaciones DB2, puede utilizar inserciones en almacenamiento intermedio en las aplicaciones, precompilando o vinculando con la opción INSERT BUF. Si la aplicación aprovecha tanto inserciones en almacenamiento como puntos de rescate, DB2 desecha el almacenamiento intermedio antes de ejecutar sentencias SAVEPOINT, RELEASE SAVEPOINT o ROLLBACK TO SAVEPOINT.

Conceptos relacionados:

- “Inserciones en almacenamiento intermedio en entornos de bases de datos particionadas” en la página 480

Consulta relacionada:

- “Mandato BIND” en el manual *Consulta de mandatos*
- “Mandato PRECOMPILE” en el manual *Consulta de mandatos*

Puntos de rescate y bloqueo de cursor

Si la aplicación utiliza puntos de rescate, considere la posibilidad de evitar el bloqueo de cursor, precompilando o vinculando la aplicación con la opción de precompilación BLOCKING NO. Aunque los cursores de bloqueo pueden mejorar el rendimiento de la aplicación al realizar una captación previa de varias filas, es posible que los datos que devuelve una aplicación que utiliza puntos de rescate y cursores de bloqueo no reflejen los datos que se han confirmado en la base de datos.

Si no precompila la aplicación con la opción BLOCKING NO, y la aplicación emite una sentencia FETCH después de que se haya producido una operación ROLLBACK TO SAVEPOINT, es posible que la sentencia FETCH recupere datos suprimidos. Por ejemplo, supongamos que la aplicación que contiene el siguiente SQL se precompila sin la opción BLOCKING NO:

```
CREATE TABLE t1(c1 INTEGER);
DECLARE CURSOR c1 AS 'SELECT c1 FROM t1 ORDER BY c1';
INSERT INTO t1 VALUES (1);
SAVEPOINT showFetchDelete;
   INSERT INTO t1 VALUES (2);
   INSERT INTO t1 VALUES (3);
   OPEN CURSOR c1;
   FETCH c1; --obtener primer valor y bloque de cursor
   ALTER TABLE t1... --añadir restricción
ROLLBACK TO SAVEPOINT;
FETCH c1; --recupera segundo valor de bloque de cursor
```

Cuando la aplicación emite la primera sentencia FETCH en la tabla “t1”, el servidor DB2 envía un bloque de valores de columna (1, 2 y 3) a la aplicación cliente. El cliente almacena localmente estos valores de columna. Cuando la aplicación emite la sentencia ROLLBACK TO SAVEPOINT SQL, los valores de columna '2' y '3' se suprimen de la tabla. Después de la sentencia ROLLBACK TO SAVEPOINT, la siguiente sentencia FETCH de la tabla devuelve el valor de columna '2', aunque dicho valor ya no existe en la tabla. La aplicación recibe este valor porque aprovecha la opción de bloqueo de cursor para mejorar el rendimiento y accede a los datos que se han almacenado localmente.

Consulta relacionada:

- “Mandato BIND” en el manual *Consulta de mandatos*
- “Mandato PRECOMPILE” en el manual *Consulta de mandatos*

Puntos de rescate y gestores de transacciones que cumplen con XA

Si hay algún punto de rescate activo en una aplicación cuando un gestor de transacciones que cumple con XA emite una petición XA_END, DB2 emite una sentencia RELEASE SAVEPOINT.

Transmisión de grandes volúmenes de datos a través de una red

Puede combinar las técnicas de procedimientos almacenados y el bloqueo de filas para mejorar significativamente el rendimiento de las aplicaciones que necesitan pasar grandes cantidades de datos a través de una red.

Las aplicaciones que pasan matrices, grandes cantidades de datos o paquetes de datos a través de la red pueden pasar los datos en bloques utilizando como mecanismo de transporte la estructura de datos SQLDA o variables del lenguaje principal. Esta técnica es sumamente potente en los lenguajes principales que soportan estructuras.

Tanto una aplicación cliente como un procedimiento de servidor pueden pasar los datos a través de la red. Los datos se pueden pasar utilizando uno de los tipos de datos siguientes:
- VARCHAR
- LONG VARCHAR
- CLOB
- BLOB

Los datos también se pueden pasar utilizando uno de los tipos de gráficos siguientes:
- VARGRAPHIC
- LONG VARGRAPHIC
- DBCLOB

Nota: cuando utilice esta técnica, asegúrese de considerar la posibilidad de convertir los caracteres. Si está pasando datos con uno de los tipos de datos de serie de caracteres, como por ejemplo VARCHAR, LONG VARCHAR o CLOB, o tipo de datos gráficos, como por ejemplo VARGRAPHIC, LONG VARGRAPHIC o DBCLOB, y si la página de códigos de la aplicación no es la misma que la de la base de datos, los datos que no sean de tipo carácter se convertirán como si lo fueran. Para evitar la conversión de caracteres, debe pasar los datos en una variable con el tipo de datos BLOB.

Conceptos relacionados:

- “Conversión de caracteres entre distintas páginas de códigos” en la página 436
- “Procedimientos almacenados de DB2” en la página 23

Tareas relacionadas:

- “Specifying row blocking to reduce overhead” en el manual *Administration Guide: Performance*

Parte 6. Apéndices

Apéndice A. Sentencias de SQL soportadas

La tabla siguiente:

- Contiene todas las sentencias de SQL soportadas en DB2 Universal Database para Linux, UNIX y sistemas operativos Windows
- Indica (mediante una 'X') si se pueden ejecutar dinámicamente
- Indica (mediante una 'X') si el procesador de línea de mandatos (CLP) las soporta
- Indica (mediante una 'X' o un nombre de función de la CLI de DB2) si se puede ejecutar la sentencia utilizando la Interfaz a nivel de llamada de DB2 (CLI de DB2)
- Indica (mediante una 'X') si se puede ejecutar la sentencia en un procedimiento de SQL

Tabla 37. Sentencias de SQL (DB2 Universal Database)

Sentencia de SQL	Dinámico[1]	Procesador de línea de mandatos (CLP)	Interfaz de nivel de llamada[3] (CLI)	Procedimiento SQL
ALLOCATE CURSOR				X
Sentencia de asignación				X
ASSOCIATE LOCATORS				X
ALTER { BUFFERPOOL, NICKNAME,[10] NODEGROUP, SERVER,[10] TABLE, TABLESPACE, USER MAPPING,[10] TYPE, VIEW }	X	X	X	
BEGIN DECLARE SECTION[2]				
CALL		X[9]	X[4]	X
Sentencia CASE				X
CLOSE		X	`SQLCloseCursor()`, `SQLFreeStmt()`	X
COMMENT ON	X	X	X	X
COMMIT	X	X	`SQLEndTran()`, `SQLTransact()`	X
SQL compuesto (incorporado)			X[4]	
Sentencia compuesta				X

Tabla 37. Sentencias de SQL (DB2 Universal Database) (continuación)

Sentencia de SQL	Dinámico[1]	Procesador de línea de mandatos (CLP)	Interfaz de nivel de llamada[3] (CLI)	Procedimiento SQL
CONNECT (Tipo 1)		X	`SQLBrowseConnect()`, `SQLConnect()`, `SQLDriverConnect()`	
CONNECT (Tipo 2)		X	`SQLBrowseConnect()`, `SQLConnect()`, `SQLDriverConnect()`	
CREATE { ALIAS, BUFFERPOOL, DISTINCT TYPE, EVENT MONITOR, FUNCTION, FUNCTION MAPPING,[10] INDEX, INDEX EXTENSION, METHOD, NICKNAME,[10] NODEGROUP, PROCEDURE, SCHEMA, SERVER, TABLE, TABLESPACE, TRANSFORM, TYPE MAPPING,[10] TRIGGER, USER MAPPING,[10] TYPE, VIEW, WRAPPER[10] }	X	X	X	X[11]
DECLARE CURSOR[2]		X	`SQLAllocStmt()`	X
DECLARE GLOBAL TEMPORARY TABLE	X	X	X	X
DELETE	X	X	X	X
DESCRIBE[8]		X	`SQLColAttributes()`, `SQLDescribeCol()`, `SQLDescribParam()`[6]	
DISCONNECT		X	`SQLDisconnect()`	
DROP	X	X	X	X[11]
END DECLARE SECTION[2]				
EXECUTE			`SQLExecute()`	X
EXECUTE IMMEDIATE			`SQLExecDirect()`	X
EXPLAIN	X	X	X	X
FETCH		X	`SQLExtendedFetch()` , `SQLFetch()`, `SQLFetchScroll()`	X
FLUSH EVENT MONITOR	X	X	X	

Tabla 37. Sentencias de SQL (DB2 Universal Database) (continuación)

Sentencia de SQL	Dinámico[1]	Procesador de línea de mandatos (CLP)	Interfaz de nivel de llamada[3] (CLI)	Procedimiento SQL
sentencia FOR				X
FREE LOCATOR			X[4]	X
GET DIAGNOSTICS				X
Sentencia GOTO				X
GRANT	X	X	X	X
Sentencia IF				X
INCLUDE[2]				
INSERT	X	X	X	X
ITERATE				X
Sentencia LEAVE				X
LOCK TABLE	X	X	X	X
Sentencia LOOP				X
OPEN		X	`SQLExecute()`, `SQLExecDirect()`	X
PREPARE			`SQLPrepare()`	X
REFRESH TABLE	X	X	X	
RELEASE		X		X
RELEASE SAVEPOINT	X	X	X	X
RENAME TABLE	X	X	X	
RENAME TABLESPACE	X	X	X	
Sentencia REPEAT				X
Sentencia RESIGNAL				X
Sentencia RETURN				X
REVOKE	X	X	X	
ROLLBACK	X	X	`SQLEndTran()`, `SQLTransact()`	X
SAVEPOINT	X	X	X	X
Sentencia-select	X	X	X	X
SELECT INTO				X
SET CONNECTION		X	`SQLSetConnection()`	
SET CURRENT DEFAULT TRANSFORM GROUP	X	X	X	X

Tabla 37. Sentencias de SQL (DB2 Universal Database) (continuación)

Sentencia de SQL	Dinámico[1]	Procesador de línea de mandatos (CLP)	Interfaz de nivel de llamada[3] (CLI)	Procedimiento SQL
SET CURRENT DEGREE	X	X	X	X
SET CURRENT EXPLAIN MODE	X	X	X, SQLSetConnectAttr()	X
SET CURRENT EXPLAIN SNAPSHOT	X	X	X, SQLSetConnectAttr()	X
SET CURRENT PACKAGESET				
SET CURRENT QUERY OPTIMIZATION	X	X	X	X
SET CURRENT REFRESH AGE	X	X	X	X
SET EVENT MONITOR STATE	X	X	X	X
SET INTEGRITY	X	X	X	
SET PASSTHRU[10]	X	X	X	X
SET PATH	X	X	X	X
SET SCHEMA	X	X	X	X
SET SERVER OPTION[10]	X	X	X	X
Variable de transición SET[5]	X	X	X	X
Sentencia SIGNAL				X
SIGNAL SQLSTATE[5]	X	X	X	
UPDATE	X	X	X	X
VALUES INTO				X
WHENEVER[2]				
Sentencia WHILE				X

Tabla 37. Sentencias de SQL (DB2 Universal Database) (continuación)

Sentencia de SQL	Dinámico[1]	Procesador de línea de mandatos (CLP)	Interfaz de nivel de llamada[3] (CLI)	Procedimiento SQL

Notas:

1. Puede codificar todas las sentencias de esta lista como SQL estático, pero sólo las marcadas con X como SQL dinámico.
2. No puede ejecutar esta sentencia.
3. Una X indica que puede ejecutar esta sentencia utilizando `SQLExecDirect()` o `SQLPrepare()` y `SQLExecute()`. Si hay una función de CLI de DB2 equivalente, se lista el nombre de la función.
4. Aunque esta sentencia no es dinámica, con CLI de DB2 puede especificar esta sentencia cuando llame a `SQLExecDirect()` o a `SQLPrepare()` y a `SQLExecute()`.
5. Sólo puede utilizarlo dentro de sentencias CREATE TRIGGER.
6. Sólo puede utilizar la sentencia de SQL DESCRIBE para describir salida, mientras que con CLI de DB2 también puede describir entrada (mediante la función `SQLDescribeParam()`).
7. Sólo puede utilizar la sentencia de SQL FETCH para captar filas de una en una en una dirección, mientras que con las funciones de CLI de DB2 `SQLExtendedFetch()` y `SQLFetchScroll()` puede captar en matrices. Además, puede captar en cualquier dirección y en cualquier posición del conjunto de resultados.
8. La sentencia de SQL DESCRIBE tiene una sintaxis diferente de la del mandato CLP DESCRIBE.
9. Cuando se emite CALL desde el procesador de línea de mandatos, sólo ciertos procedimientos y sus respectivos parámetros reciben soporte.
10. La sentencia sólo recibe soporte para servidores de bases de datos federados.
11. Los procedimientos de SQL sólo pueden emitir las sentencias CREATE y DROP para índices, tablas y vistas.

Consulta relacionada:

- "DESCRIBE sentencia" en el manual *Consulta de SQL, Volumen 2*
- "Administración de los archivos JAR en el servidor de bases de datos" en el manual *Guía de desarrollo de aplicaciones: Programación de aplicaciones de servidor*

Apéndice B. Programación en un entorno de sistema principal o iSeries

Aplicaciones en entornos de sistema principal o iSeries

DB2 Connect permite que un programa de aplicación acceda a datos de bases de datos DB2 en servidores System/390, zSeries e iSeries. Por ejemplo, una aplicación que se ejecute en Windows puede acceder a datos de una base de datos DB2 Universal Database para OS/390 y z/OS. Puede crear nuevas aplicaciones o modificar las aplicaciones existentes para que se ejecuten en un entorno de sistema principal o iSeries. También es posible desarrollar aplicaciones en un entorno y trasladarlas a otro.

DB2 Connect le permite utilizar las API siguientes con productos de base de datos de sistema principal, como por ejemplo DB2 Universal Database para OS/390 y z/OS, siempre que dichos productos soporten el elemento:

- SQL incorporado, tanto estático como dinámico
- La Interfaz a nivel de llamada de DB2
- La API ODBC de Microsoft
- JDBC

Algunas sentencias de SQL difieren según los productos de base de datos relacional. Se puede encontrar con sentencias de SQL que:

- Sean iguales para todos los productos de base de datos que utilice, independientemente de los estándares.
- Estén disponibles en todos los productos de bases de datos relacionales de IBM (vea la información de consulta de SQL para obtener más detalles).
- Sean exclusivas para el sistema de base de datos al que acceda.

La portabilidad de las sentencias de SQL de las dos primeras categorías es muy grande, pero las de la tercera categoría habrá que cambiarlas antes. En general, la portabilidad de las sentencias de SQL en Lenguaje de definición de datos (Data Definition Language - DDL) no es tan grande como la de las que están en Lenguaje de manipulación de datos (Data Manipulation Language - DML).

DB2 Connect acepta algunas sentencias de SQL que DB2 Universal Database no soporta. DB2 Connect pasa estas sentencias al servidor de sistema principal o iSeries. Para obtener información sobre los límites en las distintas plataformas, como por ejemplo la longitud máxima de columna, consulte el tema sobre límites de SQL.

Si traslada una aplicación CICS desde OS/390 o VSE para que se ejecute bajo otro producto CICS (por ejemplo, CICS para AIX), ésta también puede acceder a la base de datos OS/390 o VSE utilizando DB2 Connect. Consulte los manuales *CICS/6000 Application Programming Guide* y *CICS Customization and Operation* para obtener más detalles.

Nota: puede utilizar DB2 Connect con una base de datos DB2 Universal Database Versión 8, aunque sólo necesita un cliente DB2. La mayoría de puntos de incompatibilidad relacionados en los temas siguientes no se producirán si utiliza DB2 Connect para una base de datos DB2 Universal Database Versión 8, excepto en aquellos casos en que exista una restricción debida a una limitación del propio DB2 Connect.

Tareas relacionadas:

- "Creación de la base de datos sample en servidores de sistema principal o AS/400 e iSeries" en el manual *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

Consulta relacionada:

- "Límites de SQL" en el manual *Consulta de SQL, Volumen 1*

Lenguaje de definición de datos en entornos de sistema principal e iSeries

Las sentencias DDL difieren según el producto de base de datos IBM, puesto que el almacenamiento se maneja de forma diferente en los distintos sistemas. En los sistemas de servidor de sistema principal o iSeries, pueden existir varios pasos entre el diseño de una base de datos y la emisión de una sentencia CREATE TABLE. Por ejemplo, una serie de sentencias pueden convertir el diseño de objetos lógicos en la representación física de dichos objetos en el almacenamiento.

El precompilador pasa muchas de estas sentencias DDL al servidor de sistema principal o iSeries cuando el usuario precompila para una base de datos de servidor de sistema principal o iSeries. Las mismas sentencias no se precompilarían para una base de datos del sistema en que se esté ejecutando la aplicación. Por ejemplo, en una aplicación Windows la sentencia CREATE STORGROUP se precompilaría satisfactoriamente para una base de datos DB2 Universal Database para OS/390 y z/OS pero no para una base de datos DB2 para Windows.

Lenguaje de manipulación de datos en entornos de sistema principal e iSeries

En general, la portabilidad de las sentencias de DML es muy grande. Las sentencias SELECT, INSERT, UPDATE y DELETE son parecidas en los distintos productos de base de datos relacional de IBM. La mayoría de aplicaciones utilizan, principalmente, sentencias de SQL en DML que el programa DB2 Connect soporta.

A continuación se muestran algunas consideraciones que se deben tener en cuenta cuando se utilice DML en entornos de sistema principal e iSeries:

- Tipos de datos numéricos

  Cuando se transfieran datos numéricos a DB2 Universal Database, el tipo de datos puede cambiar. Los SQLTYPE numéricos y decimales con zona, soportados por OS/400, se convierten en SQLTYPE decimales fijos (empaquetados).

- Datos de bytes mixtos

  Los datos de bytes mixtos pueden consistir en caracteres de un juego de caracteres de código UNIX ampliado (EUC), un juego de caracteres de doble byte (DBCS) y un juego de caracteres de un solo byte (SBCS) en la misma columna. En los sistemas que almacenan los datos en EBCDIC (OS/390, z/OS, OS/400, VSE y VM), los caracteres de desplazamiento a teclado ideográfico y de desplazamiento a teclado estándar marcan el comienzo y el final de los datos de doble byte. En los sistemas que

almacenan los datos en ASCII (como UNIX), no se requieren caracteres de desplazamiento a teclado ideográfico ni de desplazamiento a teclado estándar.

Si la aplicación transfiere datos de bytes mixtos desde un sistema ASCII a un sistema EBCDIC, asegúrese de que haya suficiente espacio para los caracteres de desplazamiento de teclado. Para cada cambio de datos SBCS a DBCS, añada 2 bytes a la longitud de los datos. Para mejorar la portabilidad, utilice series de longitud variable en las aplicaciones que usen datos de bytes mixtos.

- Campos largos

  Los campos largos (series que contienen más de 254 caracteres) se manejan de forma distinta en los diferentes sistemas. Un servidor de sistema principal o iSeries sólo puede soportar un subconjunto de funciones escalares para los campos largos; por ejemplo, DB2 Universal Database para OS/390 y z/OS sólo admite las funciones **LENGTH** y **SUBSTR** para los campos largos. Asimismo, un servidor de sistema principal o iSeries puede necesitar un manejo distinto de determinadas sentencias de SQL; por ejemplo, DB2 para VSE y VM requiere que con la sentencia INSERT sólo se utilice una variable del lenguaje principal, SQLDA, o un valor NULL.

- Tipo de datos de objetos grandes

  DB2 Connect soporta el tipo de datos LOB.

- Tipos definidos por el usuario

  DB2 Connect sólo soporta los tipos diferenciados definidos por el usuario. Los tipos estructurados, también denominados tipos de datos abstractos, no reciben soporte de DB2 Connect.

- Tipo de datos ROWID

  DB2 Connect maneja el tipo de datos ROWID como VARCHAR para datos de bits.

- Tipo de datos BIGINT

  DB2 Connect soporta enteros de ocho bytes (64 bits). El tipo de datos interno BIGINT se utiliza para proporcionar soporte de cardinalidad de bases de datos muy grandes y mantener a la vez la precisión de los datos.

Lenguaje de control de datos en entornos de sistema principal e iSeries

Cada sistema de gestión de bases de datos relacionales de IBM proporciona distintos niveles de granularidad para las sentencias GRANT y REVOKE de SQL. Para verificar las sentencias de SQL adecuadas a utilizar para cada sistema de gestión de bases de datos, consulte las publicaciones específicas del producto.

Gestión de conexiones de bases de datos con DB2 Connect

DB2 Connect soporta las versiones CONNECT TO y CONNECT RESET de la sentencia CONNECT, así como CONNECT sin ningún parámetro. Si una aplicación llama a una sentencia de SQL sin antes ejecutar una sentencia CONNECT TO explícita, se realiza una conexión *implícita* del servidor de aplicaciones por omisión (si hay alguno definido).

Cuando se conecta con una base de datos, en el campo SQLERRP de la SQLCA se devuelve información que identifica el sistema de gestión de bases de datos relacionales. Si el servidor de aplicaciones es una base de datos relacional de IBM, los tres primeros bytes de SQLERRP contienen uno de los valores siguientes:

DSN DB2 Universal Database para OS/390 y z/OS

ARI DB2 para VSE y VM

QSQ DB2 UDB para iSeries

SQL DB2 Universal Database

Si se emite una sentencia CONNECT TO o CONNECT nula mientras se utiliza DB2 Connect, el código de territorio o símbolo de territorio del campo SQLERRMC de la SQLCA se devuelve en blanco; el CCSID del servidor de aplicaciones se devuelve en la señal de página de códigos o de juego de códigos.

Puede realizar explícitamente una desconexión mediante la sentencia CONNECT RESET (para la conexión de tipo 1), las sentencias RELEASE y COMMIT (para la conexión de tipo 2) o la sentencia DISCONNECT (para cualquier tipo de conexión, siempre que el entorno no sea de supervisor TP).

Nota: una aplicación puede recibir diversos SQLCODE que indiquen errores y seguir terminando normalmente; en este caso, DB2 Connect confirma los datos. Si no desea que se confirmen, debe emitir un mandato ROLLBACK.

El mandato FORCE permite desconectar de la base de datos usuarios seleccionados o todos los usuarios. Este mandato se soporta para bases de datos de servidor de sistema principal e iSeries; el usuario puede imponer una desconexión de la estación de trabajo DB2 Connect.

Consulta relacionada:

- "CONNECT (Tipo 1) sentencia" en el manual *Consulta de SQL, Volumen 2*
- "CONNECT (Tipo 2) sentencia" en el manual *Consulta de SQL, Volumen 2*

Proceso de peticiones de interrupción

DB2 Connect maneja una petición de interrupción procedente de un cliente DB2 de una de estas dos formas:

- Si aparece la palabra clave INTERRUPT_ENABLED en el campo PARMS de la entrada del catálogo DCS, DB2 Connect eliminará la conexión con el servidor de sistema principal o iSeries cuando reciba una petición de interrupción. La pérdida de conexión, al menos en servidores DB2 UDB para OS/390 y z/OS, hará que se interrumpa la petición actual en el servidor.
- Si no aparece la palabra clave INTERRUPT_ENABLED en el campo PARMS de la entrada del catálogo DCS, las peticiones de interrupción se pasan por alto.

Atributos de paquete, PREP y BIND

Las secciones siguientes describen las diferencias en los atributos de paquete entre los sistemas de bases de datos relacionales de IBM y las consideraciones a tener en cuenta al utilizar los mandatos PREPCOMPILE y BIND.

Diferencias de atributos de paquete entre sistemas de bases de datos relacionales de IBM

Un paquete tiene los atributos siguientes:

ID de colección
ID del paquete. Se puede especificar en el mandato PREP.

Propietario
ID de autorización del propietario del paquete. Se puede especificar en los mandatos PREP o BIND.

Creador
Nombre del usuario que enlaza lógicamente el paquete.

Calificador
Calificador implícito de los objetos del paquete. Se puede especificar en los mandatos PREP o BIND.

Cada sistema de servidor de sistema principal o iSeries tiene limitaciones en el uso de estos atributos:

DB2 Universal Database para OS/390 y z/OS
Los cuatro atributos pueden ser diferentes. El uso de un calificador distinto requiere privilegios de administración especiales. Para obtener más información sobre las condiciones relativas al uso de estos atributos, consulte la publicación *Command Reference* para DB2 Universal Database para OS/390 y z/OS.

DB2 para VSE y VM

Todos los atributos deben ser idénticos. Si USER1 crea un archivo de vinculación (mediante PREP) y USER2 realiza la vinculación real, USER2 necesitará autorización DBA para realizar la vinculación para USER1. Para los atributos sólo se utiliza el nombre de usuario de USER1.

DB2 UDB para iSeries

El calificador indica el nombre de colección. La relación entre calificadores y propietario afecta al otorgamiento y a la revocación de privilegios sobre el objeto. El nombre de usuario que se registra es el creador y propietario, a menos que se califique con un ID de colección, en cuyo caso el ID de colección es el propietario. El ID de colección debe existir antes de que se pueda utilizar como calificador.

DB2 Universal Database

Los cuatro atributos pueden ser diferentes. El uso de un propietario distinto requiere autorización de administración y el enlazador debe tener el privilegio CREATEIN sobre el esquema (si ya existe).

Opción CNULREQD BIND para series C terminadas en nulo

La opción de vinculación CNULREQD altera temporalmente el manejo de las series terminadas en nulo que se especifican utilizando la opción LANGLEVEL.

Por omisión, CNULREQD se establece en YES. Esto hace que las series terminadas en nulo se interpreten según los estándares MIA. Si se conecta a un servidor DB2 Universal Database para OS/390 y z/OS, es altamente recomendable establecer CNULREQD en YES. Es necesario enlazar lógicamente las aplicaciones codificadas para los estándares SAA1 (respecto a las series terminadas en nulo) con la opción CNULREQD establecida en NO. De no hacerlo, estas series se interpretarán según los estándares MIA, aunque se preparen estableciendo LANGLEVEL en SAA1.

Conceptos relacionados:

- "Series terminadas en nulo en C y C++" en la página 205

Variables SQLCODE y SQLSTATE autónomas

Las variables SQLCODE y SQLSTATE autónomas, definidas en ISO/ANS SQL92, se soportan mediante la opción de precompilación LANGLEVEL SQL92E. Durante la precompilación se emitirá un aviso SQL0020W, indicando que no se soporta LANGLEVEL. Este aviso sólo es aplicable a las características relacionadas bajo el título LANGLEVEL MIA, que es un subconjunto de LANGLEVEL SQL92E.

Consulta relacionada:

- "Mandato PRECOMPILE" en el manual *Consulta de mandatos*

Niveles de aislamiento soportados por DB2 Connect

DB2 Connect acepta los niveles de aislamiento siguientes cuando se prepara o enlaza lógicamente una aplicación:

RR Lectura repetible

RS Estabilidad de lectura

CS Estabilidad del cursor

UR Lectura no confirmada

NC Sin confirmación

Los niveles de aislamiento se relacionan en orden de mayor a menor protección. Si el servidor de sistema principal o iSeries no soporta el nivel de aislamiento que se especifique, se utiliza el próximo nivel soportado superior.

La tabla siguiente muestra el resultado de cada nivel de aislamiento en cada servidor de aplicaciones de sistema principal o iSeries.

Tabla 38. Niveles de aislamiento

DB2 Connect	DB2 Universal Database para OS/390 y z/OS	DB2 para VSE y VM	DB2 UDB para iSeries	DB2 Universal Database
RR	RR	RR	nota 1	RR
RS	nota 2	RR	COMMIT(*ALL)	RS
CS	CS	CS	COMMIT(*CS)	CS
UR	nota 3	CS	COMMIT(*CHG)	UR
NC	nota 4	nota 5	COMMIT(*NONE)	UR

Notas:

1. En DB2 UDB no existe ninguna opción COMMIT equivalente que coincida con RR. DB2 UDB para iSeries da soporte a RR bloqueando la tabla entera.
2. Resultados en RR para la Versión 3.1, y resultados en RS para la Versión 4.1 con el APAR PN75407 o para la Versión 5.1.
3. Resultados en CS para la Versión 3.1, y resultados en UR para la Versión 4.1 o para la Versión 5.1.
4. Resultados en CS para la Versión 3.1, y resultados en UR para la Versión 4.1 con el APAR PN60988 o para la Versión 5.1.
5. El nivel de aislamiento NC no se soporta con DB2 para VSE y VM.

Con DB2 UDB, puede acceder a una tabla que no sea de diario si una aplicación se enlaza con un nivel de aislamiento de UR y el bloqueo establecido en ALL, o si el nivel de aislamiento se establece en NC.

Órdenes de clasificación definidos por el usuario

Las diferencias entre EBCDIC y ASCII ocasionan diferencias en los órdenes de clasificación en los diversos productos de base de datos, y también afectan a las cláusulas ORDER BY y GROUP BY. Una manera de minimizar estas diferencias consiste en crear un orden de clasificación definido por el usuario que imite el orden de clasificación EBCDIC. Sólo se puede especificar un orden de clasificación cuando se crea una base de datos nueva.

Nota: ahora, las tablas de bases de datos se pueden almacenar en formato ASCII de DB2 Universal Database para OS/390 y z/OS. Esto permite un intercambio más rápido de los datos entre DB2 Connect y DB2 Universal Database para OS/390 y z/OS, y suprime la necesidad de proporcionar procedimientos de campo, que en lugar de esto se deberían utilizar para convertir los datos y volverlos a colocar en secuencia.

Diferencias en la integridad referencial entre sistemas de bases de datos relacionales de IBM

Los diferentes sistemas manejan las restricciones referenciales de forma distinta:

DB2 Universal Database para OS/390 y z/OS
Para poder crear una clave foránea utilizando la clave primaria, antes se debe crear un índice sobre la clave primaria. Las tablas se pueden hacer referencia a sí mismas.

DB2 para VSE y VM
Se crea automáticamente un índice para una clave foránea. Las tablas no se pueden hacer referencia a sí mismas.

DB2 UDB para iSeries
Se crea automáticamente un índice para una clave foránea. Las tablas se pueden hacer referencia a sí mismas.

DB2 Universal Database
Para las bases de datos DB2 Universal Database, se crea automáticamente un índice para una restricción exclusiva, incluida una clave primaria. Las tablas se pueden hacer referencia a sí mismas.

Existen otras normas que varían con respecto a los niveles de cascada.

Bloqueo y portabilidad de aplicaciones

La manera en que el servidor de base de datos realiza un bloqueo puede afectar a algunas aplicaciones. Por ejemplo, las aplicaciones diseñadas en base a un bloqueo de fila y aislamiento de la estabilidad del cursor no se pueden trasladar directamente a sistemas que realicen un bloqueo de página. Debido a estas diferencias fundamentales, es posible que haya que ajustar las aplicaciones.

Los productos DB2 Universal Database para OS/390 y z/OS y DB2 Universal Database ofrecen la posibilidad de marcar un tiempo de espera para un bloqueo y enviar un código de retorno de error a las aplicaciones que están a la espera.

Diferencias en SQLCODE y SQLSTATE entre sistemas de bases de datos relacionales de IBM

Distintos productos de bases de datos relacionales de IBM no siempre producen los mismos SQLCODE para errores similares. Puede manejar este problema de una de dos maneras:

- Utilice el SQLSTATE en lugar del SQLCODE para un error determinado.

  Los SQLSTATE tienen, aproximadamente, el mismo significado en los productos de base de datos, y los productos producen SQLSTATE que corresponden a los SQLCODE.
- Correlacione los SQLCODE de un sistema con los de otro.

  Por omisión, DB2 Connect correlaciona los SQLCODE y los símbolos de cada sistema de servidor de sistema principal o iSeries con el sistema DB2 Universal Database. El usuario puede especificar su propio archivo de correlación de SQLCODE si desea alterar temporalmente la correlación por omisión o está utilizando un servidor de base de datos que no dispone de correlación de SQLCODE (un servidor de base de datos no IBM). También puede desactivar la correlación de SQLCODE.

Conceptos relacionados:

- “SQLCODE mapping” en el manual *DB2 Connect User’s Guide*

Diferencias en el catálogo del sistema entre sistemas de bases de datos relacionales de IBM

Los catálogos del sistema varían en los distintos productos de base de datos de IBM. Muchas de las diferencias se pueden enmascarar mediante el uso de vistas. Para obtener información, consulte le documentación correspondiente al servidor de base de datos que utilice.

Las funciones de catálogo de la CLI evitan este problema presentando soporte de la misma API y conjuntos de resultados para las consultas de catálogos en la familia DB2.

Conceptos relacionados:

- "Catalog Functions for Querying System Catalog Information in CLI Applications" en el manual *CLI Guide and Reference, Volume 1*

Desbordamientos por conversión numérica en asignaciones de recuperación

Los distintos productos de bases de datos relacionales de IBM pueden manejar de forma distinta los desbordamientos por conversión numérica en asignaciones de recuperación. Por ejemplo, considere la operación de traer una columna de coma flotante a una variable del lenguaje principal de entero desde DB2 Universal Database para OS/390 y z/OS y desde DB2 Universal Database. Cuando se convierte el valor de coma flotante en un valor entero, se puede producir un desbordamiento por conversión. Por omisión, DB2 Universal Database para OS/390 y z/OS devolverá a la aplicación un SQLCODE de aviso y un valor nulo. En cambio, DB2 Universal Database devolverá un error de desbordamiento por conversión. Es recomendable que las aplicaciones eviten los desbordamientos por conversión numérica en las asignaciones de recuperación y realicen el movimiento a variables del lenguaje principal del tamaño adecuado.

Procedimientos almacenados en entornos de sistema principal o iSeries

Las consideraciones que deben tenerse en cuenta para procedimientos almacenados en entornos de sistema principal e iSeries son las siguientes:

- Invocación

  Un programa cliente puede invocar a un programa servidor emitiendo una sentencia CALL de SQL. En este caso, cada servidor funciona de un modo distinto a los otros servidores.

  z/OS y OS/390

  El nombre de esquema no debe tener una longitud superior a 8 bytes, el nombre de procedimiento no debe contener más de 18

bytes y el procedimiento almacenado debe estar definido en el catálogo SYSIBM.SYSPROCEDURES del servidor.

VSE o VM
El nombre de procedimiento no debe contener más de 18 bytes y debe estar definido en el catálogo SYSTEM.SYSROUTINES del servidor.

OS/400
El nombre de procedimiento debe ser un identificador de SQL. También puede utilizar las sentencias DECLARE PROCEDURE o CREATE PROCEDURE para especificar el nombre de vía de acceso real (nombre de esquema o nombre de colección) a fin de localizar el procedimiento almacenado.

Todas las sentencias CALL destinadas a DB2 UDB para iSeries desde REXX/SQL se deben preparar dinámicamente y la aplicación las tiene que ejecutar como la sentencia CALL implantada en las correlaciones de REXX/SQL con CALL USING DESCRIPTOR.

Puede invocar al programa servidor en DB2 Universal Database con el mismo convenio de parámetros que utilizan los programas servidores en DB2 Universal Database para OS/390 y z/OS, DB2 UDB para iSeries o DB2 para VSE y VM. Para obtener más información sobre el convenio de parámetros en otras plataformas, consulte la documentación del producto DB2 correspondiente a cada plataforma.

Todas las sentencias de SQL contenidas en un procedimiento almacenado se ejecutan formando parte de la unidad de trabajo SQL iniciada por el programa cliente SQL.

- No pase valores de indicador con significados especiales a los procedimientos almacenados, ni desde éstos.

  Entre DB2 Universal Database, los sistemas pasan cualquier cosa que se incluya en las variables de indicador. Sin embargo, cuando se utiliza DB2 Connect, sólo se pueden pasar 0, -1 y -128 en las variables de indicador.
- Debe definir un parámetro para devolver los posibles errores o avisos encontrados por la aplicación de servidor.

  Un programa servidor en DB2 Universal Database puede actualizar la SQLCA para devolver los posibles errores o avisos, pero un procedimiento almacenado en DB2 Universal Database para OS/390 y z/OS o DB2 UDB para iSeries no dispone de este soporte. Si desea devolver un código de error desde el procedimiento almacenado, lo debe pasar en forma de parámetro. El SQLCODE y la SQLCA sólo los establece el servidor para los errores detectados por el sistema.
- DB2 para VSE y VM Versión 7 o superior, DB2 Universal Database para OS/390 y z/OS Versión 5.1 o superior, DB2 para AS/400 V5R1 y DB2 para

iSeries Versión 7 o superior son los únicos servidores de aplicaciones de sistema principal o iSeries que actualmente pueden devolver los conjuntos de resultados de procedimientos almacenados.

Conceptos relacionados:

- "Procedimientos almacenados de DB2" en la página 23

Consulta relacionada:

- "CALL sentencia" en el manual *Consulta de SQL, Volumen 2*

Soporte de DB2 Connect de SQL compuesto

El código SQL compuesto permite que se agrupen varias sentencias de SQL en un solo bloque ejecutable. Esto puede reducir la actividad general de la red y mejorar el tiempo de respuesta.

Con SQL compuesto NOT ATOMIC, el proceso de SQL compuesto continúa después de un error. Con SQL compuesto ATOMIC, un error retrotrae el grupo entero de SQL compuesto.

Las sentencias se seguirán ejecutando hasta que el servidor de aplicaciones las termine. En general, la ejecución de la sentencia de SQL compuesto sólo se detendrá en caso de errores graves.

El código SQL compuesto NOT ATOMIC se puede utilizar con todos los servidores de aplicación de sistema principal o iSeries soportados. SQL compuesto ATOMIC se puede utilizar con servidores de aplicaciones de sistema principal soportados.

Si se producen varios errores de SQL, los SQLSTATE de las siete primeras sentencias que fallen se devuelven en el campo SQLERRMC de la SQLCA con un mensaje que indica que se han producido varios errores.

Consulta relacionada:

- "SQLCA" en el manual *Administrative API Reference*

Actualización múltiple con DB2 Connect

DB2 Connect permite realizar una actualización múltiple, también conocida como confirmación en dos fases. Una actualización múltiple es una actualización de varias bases de datos en una sola unidad de trabajo distribuida (DUOW). Si se puede o no se puede utilizar esta posibilidad, depende de varios factores:

- El programa de aplicación se debe precompilar con las opciones CONNECT 2 y SYNCPOINT TWOPHASE.
- Si tiene conexiones de red SNA, puede utilizar el soporte de confirmación en dos fases que proporciona la función de gestor de puntos de sincronismo (SPM) de DB2 Connect Enterprise Edition en AIX y Windows NT. Esta función permite a los siguientes servidores de bases de datos de sistema principal participar en una unidad de trabajo distribuida:
  - DB2 para AS/400 Versión 3.1 o posterior
  - DB2 UDB para iSeries Versión 5.1 o posterior
  - DB2 para OS/390 Versión 5.1 o posterior
  - DB2 UDB para OS/390 y z/OS Versión 7 o posterior
  - DB2 para VM & VSE Versión V5.1 o posterior.

  Esto es cierto para aplicaciones DB2 UDB nativas y aplicaciones coordinadas por un supervisor TP externo como IBM TXSeries, CICS para Open Systems, BEA Tuxedo, Encina Monitor y Microsoft Transaction Server.
- Si tiene conexiones de red TCP/IP, un servidor DB2 para OS/390 V5.1 o posteriores puede participar en una unidad de trabajo distribuida. Si la aplicación se controla mediante un Supervisor de proceso de transacciones, tal como IBM TXSeries, CICS para Open Systems, Encina Monitor o Microsoft Transaction Server, debe utilizar SPM.

  Si, tanto las aplicaciones DB2 nativas como las aplicaciones de supervisor TP, utilizan un servidor DB2 Connect Enterprise Edition común para acceder a datos del sistema principal a través de conexiones TCP/IP, se debe utilizar el gestor de puntos de sincronismo.

  Si se utiliza un solo servidor DB2 Connect Enterprise Edition para acceder a datos del sistema principal usando los dos protocolos de red SNA y TCP/IP, y si se requiere una confirmación en dos fases, debe utilizar SPM. Esto es así tanto para las aplicaciones DB2 como para las aplicaciones de supervisor TP.

Conceptos relacionados:

- "XA function supported by DB2 UDB" en el manual *Administration Guide: Planning*
- "Configuring DB2 Connect with an XA compliant transaction manager" en el manual *DB2 Connect User's Guide*

Tareas relacionadas:

- "Configuring BEA Tuxedo" en el manual *Administration Guide: Planning*
- "Updating host or iSeries database servers with an XA-compliant transaction manager" en el manual *Administration Guide: Planning*

Sentencias de SQL de servidor de sistema principal e iSeries soportadas por DB2 Connect

Las sentencias siguientes se compilan satisfactoriamente para un proceso de servidor de sistema principal o iSeries, pero no para un proceso con sistemas DB2 Universal Database:

- ACQUIRE
- DECLARE (modificador.(calificador.)nombre_tabla TABLE ...
- LABEL ON

El procesador de línea de mandatos también soporta estas sentencias.

Las sentencias siguientes se soportan para un proceso de servidor de sistema principal e iSeries, pero no se añaden al archivo de vinculación ni al paquete y el procesador de línea de mandatos no las soporta:

- DESCRIBE nombre_sentencia INTO nombre_descriptor USING NAMES
- PREPARE nombre_sentencia INTO nombre_descriptor USING NAMES FROM ...

El precompilador realiza las suposiciones siguientes:

- Las variables del lenguaje principal son variables de entrada
- Se asigna a la sentencia un número de sección exclusivo.

Sentencias de SQL de servidor de sistema principal e iSeries rechazadas por DB2 Connect

DB2 Connect y el procesador de línea de mandatos no soportan las sentencias de SQL siguientes:

- COMMIT WORK RELEASE
- DECLARE nombre_sentencia, nombre_sentencia STATEMENT
- DESCRIBE nombre_sentencia INTO nombre_descriptor USING xxxx (donde xxxx es ANY, BOTH o LABELS)
- PREPARE nombre_sentencia INTO nombre_descriptor USING xxxx FROM :variable_sist_principal (donde xxxx es ANY, BOTH o LABELS)
- PUT ...
- ROLLBACK WORK RELEASE
- SET :host_variable = CURRENT ...

Las sentencias de SQL dinámicas ampliadas de DB2 para VSE y VM se rechazan con -104 y SQLCODE de error de sintaxis.

Apéndice C. Simulación de clasificación binaria EBCDIC

Con DB2, puede clasificar series de caracteres de acuerdo con un orden de clasificación definido por el usuario. Puede utilizar esta función para simular la clasificación binaria EBCDIC.

Como ejemplo de cómo simular una clasificación EBCDIC, supongamos que desea crear una base de datos ASCII con la página de códigos 850, pero también desea que las series de caracteres se clasifiquen como si los datos realmente residieran en una base de datos EBCDIC con la página de códigos 500. Consulte las figuras siguientes para ver las definiciones de página de códigos 500 y página de códigos 850.

Supongamos la clasificación relativa de cuatro caracteres en una base de datos de página de códigos EBCDIC 500, cuando se clasifican en código binario:

Carácter	**Punto de código de página de códigos 500**
'a'	X'81'
'b'	X'82'
'A'	X'C1'
'B'	X'C2'

El orden de clasificación binaria de la página de códigos 500 (el orden deseado) es:

```
'a' < 'b' < 'A' < 'B'
```

Si crea la base de datos con la página de códigos ASCII 850, el orden de clasificación sería el siguiente:

Carácter	**Punto de código de página de códigos 850**
'a'	X'61'
'b'	X'62'
'A'	X'41'
'B'	X'42'

El orden binario de la página de códigos 850 (que no es el orden deseado) es:

```
'A' < 'B' < 'a' < 'b'
```

Para conseguir el orden deseado, tiene que crear la base de datos con un orden de clasificación definido por el usuario. Se suministra un orden de clasificación de ejemplo solo con este objetivo con DB2 en el archivo include `sqle850a.h`. El contenido de `sqle850a.h` se muestra a continuación.

```
#ifndef SQL_H_SQLE850A
#define SQL_H_SQLE850A

#ifdef __cplusplus
extern "C" {
#endif

unsigned char sqle_850_500[256] = {
0x00,0x01,0x02,0x03,0x37,0x2d,0x2e,0x2f,0x16,0x05,0x25,0x0b,0x0c,0x0d,0x0e,0x0f,
0x10,0x11,0x12,0x13,0x3c,0x3d,0x32,0x26,0x18,0x19,0x3f,0x27,0x1c,0x1d,0x1e,0x1f,
0x40,0x4f,0x7f,0x7b,0x5b,0x6c,0x50,0x7d,0x4d,0x5d,0x5c,0x4e,0x6b,0x60,0x4b,0x61,
0xf0,0xf1,0xf2,0xf3,0xf4,0xf5,0xf6,0xf7,0xf8,0xf9,0x7a,0x5e,0x4c,0x7e,0x6e,0x6f,
0x7c, 0xc1 , 0xc2 ,0xc3,0xc4,0xc5,0xc6,0xc7,0xc8,0xc9,0xd1,0xd2,0xd3,0xd4,0xd5,0xd6,
0xd7,0xd8,0xd9,0xe2,0xe3,0xe4,0xe5,0xe6,0xe7,0xe8,0xe9,0x4a,0xe0,0x5a,0x5f,0x6d,
0x79, 0x81 , 0x82 ,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x91,0x92,0x93,0x94,0x95,0x96,
0x97,0x98,0x99,0xa2,0xa3,0xa4,0xa5,0xa6,0xa7,0xa8,0xa9,0xc0,0xbb,0xd0,0xa1,0x07,
0x68,0xdc,0x51,0x42,0x43,0x44,0x47,0x48,0x52,0x53,0x54,0x57,0x56,0x58,0x63,0x67,
0x71,0x9c,0x9e,0xcb,0xcc,0xcd,0xdb,0xdd,0xdf,0xec,0xfc,0x70,0xb1,0x80,0xbf,0xff,
0x45,0x55,0xce,0xde,0x49,0x69,0x9a,0x9b,0xab,0xaf,0xba,0xb8,0xb7,0xaa,0x8a,0x8b,
0x2b,0x2c,0x09,0x21,0x28,0x65,0x62,0x64,0xb4,0x38,0x31,0x34,0x33,0xb0,0xb2,0x24,
0x22,0x17,0x29,0x06,0x20,0x2a,0x46,0x66,0x1a,0x35,0x08,0x39,0x36,0x30,0x3a,0x9f,
0x8c,0xac,0x72,0x73,0x74,0x0a,0x75,0x76,0x77,0x23,0x15,0x14,0x04,0x6a,0x78,0x3b,
0xee,0x59,0xeb,0xed,0xcf,0xef,0xa0,0x8e,0xae,0xfe,0xfb,0xfd,0x8d,0xad,0xbc,0xbe,
0xca,0x8f,0x1b,0xb9,0xb6,0xb5,0xe1,0x9d,0x90,0xbd,0xb3,0xda,0xfa,0xea,0x3e,0x41
};
#ifdef __cplusplus
}
#endif

#endif /* SQL_H_SQLE850A */
```

Figura 9. Orden de clasificación definido por el usuario - sqle_850_500

Para ver cómo conseguir el orden de clasificación de la página de códigos 500 en caracteres de la página de códigos 850, observe el orden de clasificación de ejemplo de sqle_850_500. Para cada carácter de la página de códigos 850, su peso en el orden de clasificación es simplemente su punto de código correspondiente en la página de códigos 500.

Por ejemplo, supongamos que tenemos la letra ‘a’. Esta letra tiene el punto de código X'61' para la página de códigos 850. En la matriz sqle_850_500, a la letra ‘a’ se le asigna un peso de X'81' (es decir, el elemento número 98 de la matriz sqle_850_500).

Veamos cómo se clasifican los cuatro caracteres cuando se crea la base de datos con el orden de clasificación definido por el usuario del ejemplo anterior:

Carácter	Punto código página códigos 850 / Peso (de sqle_850_500)
'a'	X'61' / X'81'
'b'	X'62' / X'82'
'A'	X'41' / X'C1'
'B'	X'42' / X'C2'

El orden definido por el usuario de la página de códigos 850 por peso (el orden deseado) es:

```
'a' < 'b' < 'A' < 'B'
```

En este ejemplo, consigue el orden deseado especificando los pesos correctos para simular el comportamiento deseado.

Mirando detenidamente el orden de clasificación real, se observa que el orden en sí es simplemente una tabla de conversión, en la que la página de códigos fuente es la página de códigos de la base de datos (850) y la página de códigos de destino es la página de códigos del orden binario deseado (500). Otros órdenes de clasificación de ejemplo que se suministran con DB2 permiten otras conversiones. Si una tabla de conversión que necesita no se suministra con DB2, puede obtener tablas de conversión adicionales de la publicación de IBM *Character Data Representation Architecture, Reference and Registry*, SC09-2190. Encontrará las tablas de conversión adicionales en un CD-ROM que se suministra con dicha publicación.

DIGIT HEX 1° → 2° ↓	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0	(SP) SP010000	& SM030000	- SP100000	ø LO610000	Ø LO620000	° SM190000	µ SM170000	¢ SC040000	{ SM110000	} SM140000	\ SM070000	0 ND100000
-1	(RSP) SP300000	é LE110000	/ SP120000	É LE120000	a LA010000	j LJ010000	~ SD190000	£ SC020000	A LA020000	J LJ020000	÷ SA060000	1 ND010000
-2	â LA150000	ê LE150000	Â LA160000	Ê LE160000	b LB010000	k LK010000	s LS010000	¥ SC050000	B LB020000	K LK020000	S LS020000	2 ND020000
-3	ä LA170000	ë LE170000	Ä LA180000	Ë LE180000	c LC010000	l LL010000	t LT010000	· SD630000	C LC020000	L LL020000	T LT020000	3 ND030000
-4	à LA130000	è LE130000	À LA140000	È LE140000	d LD010000	m LM010000	u LU010000	© SM520000	D LD020000	M LM020000	U LU020000	4 ND040000
-5	á LA110000	í LI110000	Á LA120000	Í LI120000	e LE010000	n LN010000	v LV010000	§ SM240000	E LE020000	N LN020000	V LV020000	5 ND050000
-6	ã LA190000	î LI150000	Ã LA200000	Î LI160000	f LF010000	o LO010000	w LW010000	¶ SM250000	F LF020000	O LO020000	W LW020000	6 ND060000
-7	å LA270000	ï LI170000	Å LA280000	Ï LI180000	g LG010000	p LP010000	x LX010000	¼ NF040000	G LG020000	P LP020000	X LX020000	7 ND070000
-8	ç LC410000	ì LI130000	Ç LC420000	Ì LI140000	h LH010000	q LQ010000	y LY010000	½ NF010000	H LH020000	Q LQ020000	Y LY020000	8 ND080000
-9	ñ LN190000	ß LS610000	Ñ LN200000	` SD130000	i LI010000	r LR010000	z LZ010000	¾ NF050000	I LI020000	R LR020000	Z LZ020000	9 ND090000
-A	[SM060000	] SM080000	 SM650000	: SP130000	« SP170000	ª SM210000	¡ SP030000	¬ SM660000	¯ (SHY) SP320000	¹ ND011000	² ND021000	³ ND031000
-B	. SP110000	$ SC030000	, SP080000	# SM010000	» SP180000	º SM200000	¿ SP160000	Ì SM130000	ô LO150000	û LU150000	Ô LO160000	Û LU160000
-C	< SA030000	* SM040000	% SM020000	@ SM050000	ð LD630000	æ LA510000	Ð LD620000	¯ SM150000	ö LO170000	ü LU170000	Ö LO180000	Ü LU180000
-D	(SP060000	) SP070000	_ SP090000	' SP050000	ý LY110000	, SD410000	Ý LY120000	¨ SD170000	ò LO130000	ù LU130000	Ò LO140000	Ù LU140000
-E	+ SA010000	; SP140000	> SA050000	= SA040000	þ LT630000	Æ LA520000	Þ LT640000	´ SD110000	ó LO110000	ú LU110000	Ó LO120000	Ú LU120000
-F	! SP020000	^ SD150000	? SP150000	" SP040000	± SA020000	¤ SC010000	® SM530000	× SA070000	õ LO190000	ÿ LY170000	Õ LO200000	 (EO)

Página de codigos 00500

Figura 10. Página de códigos 500

DIGITS HEX 1° → 2° ↓	0-	1-	2-	3-	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0		► SM590000	(SP) SP010000	0 ND100000	@ SM050000	P LP020000	` SD130000	p LP010000	Ç LC420000	É LE120000	á LA110000	▒ SF140000	┼ SF020000	ð LD630000	Ó LO120000	¯ (SHY) SP320000
-1	☺ SS000000	◄ SM630000	! SP020000	1 ND010000	A LA020000	Q LQ020000	a LA010000	q LQ010000	ü LU170000	æ LA510000	í LI110000	▓ SF150000	╨ SF070000	Đ LD620000	ß LS610000	± SA020000
-2	☻ SS010000	↕ SM760000	" SP040000	2 ND020000	B LB020000	R LR020000	b LB010000	r LR010000	é LE110000	Æ LA520000	ó LO110000	▓ SF160000	╥ SF060000	Ê LE160000	Ô LO160000	═ SM100000
-3	♥ SS020000	!! SP330000	# SM010000	3 ND030000	C LC020000	S LS020000	c LC010000	s LS010000	â LA150000	ô LO150000	ú LU110000	╢ SF110000	╪ SF080000	Ë LE180000	Ò LO140000	¾ NF050000
-4	♦ SS030000	¶ SM250000	$ SC030000	4 ND040000	D LD020000	T LT020000	d LD010000	t LT010000	ä LA170000	ö LO170000	ñ LN190000	╡ SF090000	╞ SF100000	È LE140000	õ LO190000	¶ SM250000
-5	♣ SS040000	§ SM240000	% SM020000	5 ND050000	E LE020000	U LU020000	e LE010000	u LU010000	à LA130000	ò LO130000	Ñ LN200000	Á LA120000	╫ SF050000	¹ LI610000	Õ LO200000	§ SM240000
-6	♠ SS050000	▬ SM700000	& SM030000	6 ND060000	F LF020000	V LV020000	f LF010000	v LV010000	å LA270000	û LU150000	ª SM210000	Â LA160000	ã LA190000	Í LI120000	µ SM170000	÷ SA060000
-7	• SM570000	↨ SM770000	' SP050000	7 ND070000	G LG020000	W LW020000	g LG010000	w LW010000	ç LC410000	ù LU130000	º SM200000	À LA140000	Ã LA200000	Î LI160000	þ LT630000	¸ SD410000
-8	◘ SM570001	↑ SM320000	(SP060000	8 ND080000	H LH020000	X LX020000	h LH010000	x LX010000	ê LE150000	ÿ LY170000	¿ SP160000	© SM520000	╕ SF380000	Ï LI180000	Þ LT640000	° SM190000
-9	○ SM750000	↓ SM330000	) SP070000	9 ND090000	I LI020000	Y LY020000	i LI010000	y LY010000	ë LE170000	Ö LO180000	® SM530000	╣ SF230000	╒ SF390000	╬ SF040000	Ú LU120000	¨ SD170000
-A	◙ SM750002	→ SM310000	* SM040000	: SP130000	J LJ020000	Z LZ020000	j LJ010000	z LZ010000	è LE130000	Ü LU180000	¬ SM660000	╚ SF240000	╓ SF400000	╧ SF010000	Û LU160000	• SD630000
-B	♂ SM280000	← SM300000	+ SA010000	; SP140000	K LK020000	[SM060000	k LK010000	{ SM110000	ï LI170000	ø LO610000	½ NF010000	╗ SF250000	╫ SF410000	▌ SF610000	Ù LU140000	1 ND011000
-C	♀ SM290000	└ SA420000	, SP080000	< SA030000	L LL020000	\ SM070000	l LL010000	| SM130000	î LI150000	£ SC020000	¼ NF040000	▀ SF260000	╙ SF420000	▄ SF570000	ý LY110000	3 ND031000
-D	♪ SM930000	↔ SM780000	- SP100000	= SA040000	M LM020000	] SM080000	m LM010000	} SM140000	ì LI130000	Ø LO620000	¡ SP030000	¢ SC040000	╤ SF430000	SM650000	Ý LY120000	2 ND021000
-E	♫ SM910000	▲ SM600000	. SP110000	> SA050000	N LN020000	^ SD150000	n LN010000	~ SD190000	Ä LA180000	× SA070000	« SP170000	¥ SC050000	╬ SF440000	Ì LI140000	¯ SM150000	■ SM470000
-F	☼ SM690000	▼ SV040000	/ SP120000	? SP150000	O LO020000	_ SP090000	o LO010000	⌂ SM790000	Å LA280000	ƒ SC070000	» SP180000	╧ SF030000	¤ SC010000	▄ SF600000	´ SD110000	(RSP) SP300000

Página de códigos 00850

Figura 11. Página de códigos 850

Conceptos relacionados:

- “Órdenes de clasificación” en la página 422

Consulta relacionada:

- “sqlecrea - Create Database” en el manual *Administrative API Reference*

Índice

Caracteres Especiales

A

B

C

D

E

F

G

H

I

J

L

M

Q

R

S

T

U

V

W

X

Información técnica sobre DB2 Universal Database

Visión general de la información técnica de DB2 Universal Database

La información técnica de DB2 Universal Database puede obtenerse en los formatos siguientes:

- Manuales (formatos PDF y copia impresa)
- Un árbol de temas (formato HTML)
- Herramientas de ayuda para DB2 (formato HTML)
- Programas de ejemplo (formato HTML)
- Ayuda de línea de mandatos
- Guías de aprendizaje

Esta sección es una visión general de la información técnica que se proporciona y del modo en que se puede acceder a ella.

FixPaks para la documentación de DB2

IBM puede poner periódicamente a disposición del usuario FixPaks de documentación. Los FixPaks de documentación permiten actualizar la información que se instaló desde el *CD de documentación HTML de DB2* a medida que aparece nueva información.

Nota: Si instala los FixPaks de documentación, la documentación HTML contendrá información más reciente que los manuales de DB2 en formato PDF en línea o impresos.

Categorías de la información técnica de DB2

La información técnica de DB2 se clasifica por categorías con las cabeceras siguientes:

- Información básica de DB2
- Información de administración
- Información para el desarrollo de aplicaciones
- Información de Inteligencia empresarial
- Información de DB2 Connect
- Información de iniciación
- Información de aprendizaje
- Información sobre componentes opcionales
- Notas del release

Las tablas siguientes describen, para cada manual de la biblioteca de DB2, la información necesaria para solicitar la copia impresa, imprimir o ver el PDF o localizar el directorio de HTML de dicho manual. En el Centro de publicaciones de IBM de la dirección www.ibm.com/shop/publications/order se encuentra disponible una descripción completa de cada uno de los manuales de la biblioteca de DB2.

El directorio de instalación del CD de documentación HTML es diferente para cada categoría de información:

víaaccesocdhtml/doc/htmlcd/*%L*/*categoría*

donde:

- *víaaccesocdhtml* es el directorio donde está instalado el CD de HTML.
- *%L* es el identificador de idioma. Por ejemplo, es_ES.
- *categoría* es el identificador de categoría. Por ejemplo, core para la información básica de DB2.

En la columna de nombre de archivo PDF de las tablas siguientes, el carácter situado en la sexta posición del nombre de archivo indica la versión de idioma de un manual. Por ejemplo, el nombre de archivo db2d1e80 identifica la versión inglesa del manual *Administration Guide: Planning* y el nombre de archivo db2d1g80 identifica la versión alemana del mismo manual. En la sexta posición de los nombres de archivo se utilizan las letras siguientes para indicar el idioma del manual:

Idioma	**Identificador**
Alemán	g
Búlgaro	u
Checo	x
Chino simplificado	c
Chino tradicional	t
Coreano	k
Croata	9
Danés	d
Eslovaco	7
Esloveno	l
Español	z
Finlandés	y
Francés	f
Griego	a
Holandés	q
Húngaro	h
Inglés	e
Italiano	i
Japonés	j

Noruego	n
Polaco	p
Portugués de Brasil	b
Portugués	v
Rumano	8
Ruso	r
Sueco	s
Turco	m
Árabe	w

Sin número de documento indica que el manual sólo está disponible en línea y no tiene una versión impresa.

Información básica de DB2

La información de esta categoría incluye temas de DB2 que son fundamentales para todos los usuarios de DB2. Encontrará útil la información de esta categoría tanto si es programador o administrador de bases de datos como si trabaja con DB2 Connect, DB2 Warehouse Manager u otros productos DB2.

El directorio de instalación de esta categoría es doc/htmlcd/*%L*/core.

Tabla 39. Información básica de DB2

Nombre	Número de documento	Nombre de archivo PDF
IBM DB2 Universal Database Consulta de mandatos	SC10-3725	db2n0x80
IBM DB2 Universal Database Glosario	Sin número de documento	db2t0x80
IBM DB2 Universal Database Master Index	SC09-4839	db2w0x80
IBM DB2 Universal Database Consulta de mensajes, Volumen 1	GC10-3728	db2m1x80
IBM DB2 Universal Database Consulta de mensajes, Volumen 2	GC10-3729	db2m2x80
IBM DB2 Universal Database Novedades	SC10-3734	db2q0x80

Información de administración

La información de esta categoría incluye los temas necesarios para diseñar, implementar y mantener de forma efectiva bases de datos de DB2, depósitos de datos y sistemas federados.

El directorio de instalación de esta categoría es doc/htmlcd/%L/admin.

Tabla 40. Información de administración

Nombre	Número de documento	Nombre de archivo PDF
IBM DB2 Universal Database Administration Guide: Planning	SC09-4822	db2d1x80
IBM DB2 Universal Database Administration Guide: Implementation	SC09-4820	db2d2x80
IBM DB2 Universal Database Administration Guide: Performance	SC09-4821	db2d3x80
IBM DB2 Universal Database Administrative API Reference	SC09-4824	db2b0x80
IBM DB2 Universal Database Data Movement Utilities Guide and Reference	SC09-4830	db2dmx80
IBM DB2 Universal Database Data Recovery and High Availability Guide and Reference	SC09-4831	db2hax80
IBM DB2 Universal Database Data Warehouse Center Administration Guide	SC27-1123	db2ddx80
IBM DB2 Universal Database Federated Systems Guide	GC27-1224	db2fpx80
IBM DB2 Universal Database Guía de las herramientas de la GUI para la administración y el desarrollo	SC10-3732	db2atx80
IBM DB2 Universal Database Replication Guide and Reference	SC27-1121	db2e0x80
IBM DB2 Instalación y administración de un entorno de satélites	GC10-3770	db2dsx80
IBM DB2 Universal Database Consulta de SQL, Volumen 1	SC10-3730	db2s1x80
IBM DB2 Universal Database Consulta de SQL, Volumen 2	SC10-3731	db2s2x80
IBM DB2 Universal Database System Monitor Guide and Reference	SC09-4847	db2f0x80

Información para el desarrollo de aplicaciones

La información de esta categoría es de especial interés para los programadores de aplicaciones o programadores que trabajan con DB2. Encontrará información acerca de los lenguajes y compiladores soportados, así como la documentación necesaria para acceder a DB2 utilizando las diversas interfaces de programación soportadas, por ejemplo SQL incorporado, ODBC, JDBC, SQLj y CLI. Si visualiza esta información en línea en HTML, también podrá acceder a un conjunto de programas de DB2 de ejemplo en HTML.

El directorio de instalación de esta categoría es doc/htmlcd/*%L*/ad.

Tabla 41. Información para el desarrollo de aplicaciones

Nombre	Número de documento	Nombre de archivo PDF
IBM DB2 Universal Database Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones	SC10-3733	db2axx80
IBM DB2 Universal Database Guía de desarrollo de aplicaciones: Programación de aplicaciones de cliente	SC10-3723	db2a1x80
IBM DB2 Universal Database Guía de desarrollo de aplicaciones: Programación de aplicaciones de servidor	SC10-3724	db2a2x80
IBM DB2 Universal Database Call Level Interface Guide and Reference, Volume 1	SC09-4849	db2l1x80
IBM DB2 Universal Database Call Level Interface Guide and Reference, Volume 2	SC09-4850	db2l2x80
IBM DB2 Universal Database Data Warehouse Center Application Integration Guide	SC27-1124	db2adx80
IBM DB2 XML Extender Administración y programación	SC10-3750	db2sxx80

Información de inteligencia empresarial

La información de esta categoría describe cómo utilizar los componentes que mejoran las posibilidades de análisis y de depósito de datos de DB2 Universal Database.

El directorio de instalación de esta categoría es doc/htmlcd/*%L*/wareh.

Tabla 42. Información de Inteligencia empresarial

Nombre	Número de documento	Nombre de archivo PDF
IBM DB2 Warehouse Manager Information Catalog Center Administration Guide	SC27-1125	db2dix80
IBM DB2 Warehouse Manager Guía de instalación	GC10-3746	db2idx80

Información de DB2 Connect

La información de esta categoría describe cómo acceder a los datos de sistema principal o de iSeries utilizando DB2 Connect Enterprise Edition o DB2 Connect Personal Edition.

El directorio de instalación de esta categoría es doc/htmlcd/*%L*/conn.

Tabla 43. Información de DB2 Connect

Nombre	Número de documento	Nombre de archivo PDF
APPC, CPI-C, and SNA Sense Codes	Sin número de documento	db2apx80
IBM Connectivity Supplement	Sin número de documento	db2h1x80
IBM DB2 Connect Guía rápida de iniciación para DB2 Enterprise Edition	GC10-3774	db2c6x80
IBM DB2 Connect Quick Beginnings for DB2 Connect Personal Edition	GC09-4834	db2c1x80
IBM DB2 Connect User's Guide	SC09-4835	db2c0x80

Información de iniciación

La información de esta categoría es útil cuando se van a instalar y configurar servidores, clientes y otros productos de DB2.

El directorio de instalación de esta categoría es doc/htmlcd/%L/start.

Tabla 44. Información de iniciación

Nombre	Número de documento	Nombre de archivo PDF
IBM DB2 Universal Database Guía rápida de iniciación para clientes de DB2	GC10-3775	db2itx80
IBM DB2 Universal Database Guía rápida de iniciación para servidores de DB2	GC10-3773	db2isx80
IBM DB2 Universal Database Guía rápida de iniciación para DB2 Personal Edition	GC10-3771	db2i1x80
IBM DB2 Universal Database Suplemento de instalación y configuración	GC10-3772	db2iyx80
IBM DB2 Universal Database Guía rápida de iniciación para DB2 Data Links Manager	GC10-3726	db2z6x80

Información de aprendizaje

La información de aprendizaje presenta las características de DB2 y explica cómo realizar diversas tareas.

El directorio de instalación de esta categoría es doc/htmlcd/%L/tutr.

Tabla 45. Información de aprendizaje

Nombre	Número de documento	Nombre de archivo PDF
Guía de aprendizaje de Inteligencia empresarial: Introducción al depósito de datos	Sin número de documento	db2tux80
Guía de aprendizaje de Inteligencia empresarial: Lecciones ampliadas sobre el depósito de datos	Sin número de documento	db2tax80
Development Center Tutorial for Video Online using Microsoft Visual Basic	Sin número de documento	db2tdx80

Tabla 45. Información de aprendizaje (continuación)

Nombre	Número de documento	Nombre de archivo PDF
Information Catalog Center Tutorial	Sin número de documento	db2aix80
Guía de aprendizaje de Video Central para e-business	Sin número de documento	db2twx80
Guía de aprendizaje de Visual Explain	Sin número de documento	db2tvx80

Información sobre componentes opcionales

La información de esta categoría describe cómo trabajar con los componentes opcionales de DB2.

El directorio de instalación de esta categoría es doc/htmlcd/*%L*/opt.

Tabla 46. Información sobre componentes opcionales

Nombre	Número de documento	Nombre de archivo PDF
IBM DB2 Life Sciences Data Connect Guía de planificación, instalación y configuración	SC10-3747	db2lsx80
IBM DB2 Spatial Extender Guía del usuario y de consulta	SC10-3755	db2sbx80
IBM DB2 Universal Database Data Links Manager Administration Guide and Reference	SC27-1221	db2z0x80
IBM DB2 Universal Database Net Search Extender Guía de administración y programación **Nota:** El HTML para este documento no se instala desde el CD de documentación HTML.	SH10-9305	N/D

Notas del release

Las notas del release proporcionan información adicional específica del release y nivel de FixPak del producto. También proporcionan resúmenes de las actualizaciones de la documentación que se han incorporado en cada release y FixPak.

Tabla 47. Notas del release

Nombre	Número de documento	Nombre de archivo PDF
Notas del release de DB2	Ver nota.	Ver nota.
Notas de instalación de DB2	Sólo disponible en el CD-ROM del producto.	Sólo disponible en el CD-ROM del producto.

Nota: La versión HTML de las notas del release está disponible en el Centro de información y en los CD-ROM del producto. Para ver el archivo ASCII en plataformas basadas en UNIX, consulte el archivo Release.Notes. Este archivo se encuentra en el directorio DB2DIR/Readme/*%L*, donde *%L* representa el nombre de entorno nacional y DB2DIR representa:

- /usr/opt/db2_08_01 en AIX
- /opt/IBM/db2/V8.1 en todos los demás sistemas operativos UNIX

Tareas relacionadas:

- "Impresión de manuales de DB2 desde archivos PDF" en la página 581
- "Solicitud de manuales de DB2 impresos" en la página 582
- "Acceso a la ayuda en línea" en la página 583
- "Búsqueda de información de productos mediante el acceso al Centro de información de DB2 desde las herramientas de administración" en la página 587
- "Cómo ver documentación técnica en línea directamente desde el CD de documentación HTML de DB2" en la página 589

Impresión de manuales de DB2 desde archivos PDF

Puede imprimir los manuales de DB2 desde los archivos PDF del *CD de documentación PDF de DB2*. Mediante la utilización de Adobe Acrobat Reader, puede imprimir el manual entero o un rango específico de páginas.

Prerrequisitos:

Asegúrese de tener Adobe Acrobat Reader. Está disponible en el sitio Web de Adobe en www.adobe.com

Procedimiento:

Para imprimir un manual de DB2 desde un archivo PDF:

1. Inserte el *CD de documentación PDF de DB2*. En sistemas operativos UNIX, monte el CD de documentación PDF de DB2. Consulte el manual *Iniciación rápida* para obtener detalles sobre cómo montar un CD en sistemas operativos UNIX.
2. Inicie Adobe Acrobat Reader.
3. Abra un archivo PDF desde una de las ubicaciones siguientes:
   - En sistemas operativos Windows:

     x:\doc*idioma*, donde *x* representa la letra de unidad del CD-ROM e *idioma* representa el código de territorio de dos caracteres que representa el idioma (por ejemplo, EN para inglés).
   - En sistemas operativos UNIX:

     /cdrom/doc/*%L* del CD-ROM, donde */cdrom* representa el punto de montaje del CD-ROM y *%L* representa el entorno nacional deseado.

Tareas relacionadas:

- "Solicitud de manuales de DB2 impresos" en la página 582
- "Búsqueda de información de productos mediante el acceso al Centro de información de DB2 desde las herramientas de administración" en la página 587
- "Cómo ver documentación técnica en línea directamente desde el CD de documentación HTML de DB2" en la página 589

Consulta relacionada:

- "Visión general de la información técnica de DB2 Universal Database" en la página 573

Solicitud de manuales de DB2 impresos

Procedimiento:

Para solicitar manuales impresos:

- Póngase en contacto con el distribuidor autorizado o representante de márketing de IBM. Para encontrar un representante local de IBM, consulte el directorio mundial de contactos de IBM en la página Web www.ibm.com/planetwide
- Llame al teléfono 1-800-879-2755, si está en los EE.UU. o al 1-800-IBM-4YOU, si está en Canadá.
- Visite el Centro de publicaciones de IBM en www.ibm.com/shop/publications/order

También puede obtener manuales de DB2 impresos si solicita los Doc Pack para el producto DB2 a su distribuidor de IBM. Los Doc Pack son

subconjuntos de los manuales de la biblioteca de DB2 seleccionados con el objeto de ayudar al usuario a empezar a utilizar el producto DB2 que ha adquirido. Los manuales de los Doc Pack son los mismos que los que se encuentran en formato PDF en el *CD de documentación PDF de DB2* y presentan el mismo contenido que la documentación que se encuentra en el *CD de documentación HTML de DB2*.

Tareas relacionadas:

- "Impresión de manuales de DB2 desde archivos PDF" en la página 581
- "Búsqueda de temas mediante el acceso al Centro de información de DB2 desde un navegador" en la página 585
- "Cómo ver documentación técnica en línea directamente desde el CD de documentación HTML de DB2" en la página 589

Consulta relacionada:

- "Visión general de la información técnica de DB2 Universal Database" en la página 573

Acceso a la ayuda en línea

La ayuda en línea que viene con todos los componentes de DB2 está disponible en tres tipos:

- Ayuda de ventana y de cuaderno
- Ayuda de línea de mandatos
- Ayuda de sentencia de SQL

La ayuda de ventana y de cuaderno explica las tareas que se pueden realizar en una ventana o un cuaderno y describe los controles. Esta ayuda tiene dos tipos:

- Ayuda accesible desde el botón **Ayuda**
- Ventanas emergentes de información

El botón **Ayuda** proporciona acceso a la información de visión general y de prerrequisitos. Las ventanas emergentes de información describen los controles de la ventana o del cuaderno. La ayuda de ventana y de cuaderno está disponible en los centros y componentes de DB2 que tienen interfaces de usuario.

La ayuda de línea de mandatos incluye ayuda de mandatos y ayuda de mensajes. La ayuda de mandatos explica la sintaxis de los mandatos del procesador de línea de mandatos. La ayuda de mensajes describe la causa de un mensaje de error y describe la acción que se debe realizar en respuesta al error.

La ayuda de sentencia de SQL incluye la ayuda de SQL y la ayuda de SQLSTATE. DB2 devuelve un valor de SQLSTATE para las condiciones que pueden ser el resultado de una sentencia de SQL. La ayuda de SQLSTATE explica la sintaxis de las sentencias de SQL (códigos de clase y estados de SQL).

Nota: Para los sistemas operativos UNIX no hay ayuda de SQL disponible.

Procedimiento:

Para acceder a la ayuda en línea:

- Para la ayuda de ventana y cuaderno, pulse **Ayuda** o pulse dicho control y, a continuación, pulse **F1**. Si se selecciona el recuadro de selección **Visualizar automáticamente ventanas emergentes de información** en la página **General** del cuaderno **Valores de herramientas**, también podrá ver la ventana emergente de información para un control determinado manteniendo el cursor del ratón sobre el control.
- Para la ayuda de línea de mandatos, abra el procesador de línea de mandatos y entre:
  - Para la ayuda de mandatos:

    ```
    ? mandato
    ```

    donde *mandato* representa una palabra clave o el mandato completo.

    Por ejemplo, `? catalog` visualiza la ayuda para todos los mandatos CATALOG, mientras que `? catalog database` visualiza la ayuda para el mandato CATALOG DATABASE.
- Para la ayuda de mensajes:

  ```
  ? XXXnnnnn
  ```

  donde *XXXnnnnn* representa un identificador de mensaje válido.

  Por ejemplo, `? SQL30081` visualiza la ayuda acerca del mensaje SQL30081.
- Para la ayuda de sentencia 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, mientras que `? 08` visualiza la ayuda para el código de clase 08.

Tareas relacionadas:

- “Búsqueda de temas mediante el acceso al Centro de información de DB2 desde un navegador” en la página 585
- “Cómo ver documentación técnica en línea directamente desde el CD de documentación HTML de DB2” en la página 589

Búsqueda de temas mediante el acceso al Centro de información de DB2 desde un navegador

Si accede al Centro de información de DB2 desde un navegador podrá acceder a la información que necesita para obtener el máximo provecho de DB2 Universal Database y DB2 Connect. El Centro de información de DB2 también documenta las características y los componentes principales de DB2, entre ellos, la duplicación, el almacenamiento de datos, los metadatos y DB2 Extenders.

El Centro de información de DB2 al que se accede desde un navegador se compone de los siguientes elementos principales:

Árbol de navegación
El árbol de navegación está ubicado en el marco izquierdo de la ventana del navegador. El árbol se expande y se contrae para mostrar y ocultar los temas, el glosario y el índice maestro del Centro de información de DB2.

Barra de herramientas de navegación
La barra de herramientas de navegación está ubicada en el marco superior derecho de la ventana del navegador. La barra de herramientas de navegación contiene botones que permiten realizar búsquedas en el Centro de información de DB2, ocultar el árbol de navegación y buscar el tema visualizado actualmente en el árbol de navegación.

Marco de contenido
El marco de contenido está ubicado en el marco inferior derecho de la ventana del navegador. El marco de contenido visualiza los temas del Centro de información de DB2 cuando se pulsa un enlace en el árbol de navegación, se pulsa un resultado de búsqueda o se sigue un enlace desde otro tema o desde el índice maestro.

Prerrequisitos:

Para acceder al Centro de información de DB2 desde un navegador, deberá utilizar uno de los navegadores siguientes:

- Microsoft Explorer, versión 5 o posterior
- Netscape Navigator, versión 6.1 o posterior

Restricciones:

El Centro de información de DB2 sólo contiene los conjuntos de temas que se elige instalar desde el *CD de documentación HTML de DB2*. Si el navegador Web devuelve un error `Archivo no encontrado` cuando se intenta seguir un enlace a un tema, habrá que instalar uno o varios conjuntos de temas adicionales del *CD de documentación HTML de DB2*.

Procedimiento:

Para encontrar un tema buscándolo mediante palabras clave:

1. En la barra de herramientas de navegación, pulse **Buscar**.
2. En el campo de entrada de texto superior de la ventana Buscar, escriba uno o más términos relacionados con el área de interés, y pulse **Buscar**. Se visualiza una lista de temas clasificados por exactitud en el campo **Resultados**. La clasificación numérica junto al acierto proporciona una indicación de la importancia de la coincidencia (los números más altos indican resultados coincidentes más importantes).

   La entrada de más términos aumentará la precisión de la consulta al mismo tiempo que reducirá el número de temas devueltos.
3. En el campo **Resultados**, pulse el título del tema que desea leer. El tema se visualiza en el marco de contenido.

Para buscar un tema en el árbol de navegación:

1. En el árbol de navegación, pulse el icono de libro de la categoría de temas relacionados con el área de interés. Se visualiza una lista de subcategorías bajo el icono.
2. Continúe pulsando los iconos de libro hasta que encuentre la categoría que contiene los temas en los que está interesado. Las categorías que enlazan con temas visualizan el título de categoría como un enlace subrayado al mover el cursor sobre el título de categoría. El árbol de navegación identifica los temas con un icono de página.
3. Pulse el enlace al tema. El tema se visualiza en el marco de contenido.

Para buscar un tema o un término en el índice maestro:

1. En el árbol de navegación, pulse la categoría “Índice”. La categoría se expande para visualizar una lista de enlaces ordenados alfabéticamente en el árbol de navegación.
2. En el árbol de navegación, pulse el enlace correspondiente al primer carácter del término relacionado con el tema en el que está interesado. En el marco de contenido se visualiza una lista de términos con dicho carácter inicial. Los términos que tienen varias entradas de índice se identifican mediante un icono de libro.

3. Pulse el icono de libro correspondiente al término en el que está interesado. Se visualiza una lista de subtérminos y temas debajo del término que ha pulsado. Los temas se identifican mediante iconos de página con un título subrayado.
4. Pulse el título del tema que satisface sus necesidades. El tema se visualiza en el marco de contenido.

Conceptos relacionados:

- "Accesibilidad" en la página 595
- "Acceso al Centro de información de DB2 desde un navegador" en la página 597

Tareas relacionadas:

- "Búsqueda de información de productos mediante el acceso al Centro de información de DB2 desde las herramientas de administración" en la página 587
- "Actualización de la documentación HTML instalada en la máquina" en la página 590
- "Resolución de problemas de búsqueda de documentación de DB2 con Netscape 4.x" en la página 592
- "Búsqueda en la documentación de DB2" en la página 593

Consulta relacionada:

- "Visión general de la información técnica de DB2 Universal Database" en la página 573

Búsqueda de información de productos mediante el acceso al Centro de información de DB2 desde las herramientas de administración

El Centro de información de DB2 proporciona acceso rápido a la información de productos de DB2 y está disponible en todos los sistemas operativos para los que están disponibles las herramientas de administración de DB2.

El Centro de información de DB2 al que se accede desde las herramientas proporciona seis tipos de información.

Tareas Tareas clave que puede realizar mediante DB2.

Conceptos
Conceptos clave para DB2.

Consulta
Información de consulta de DB2, tal como palabras clave, mandatos y las API.

Resolución de problemas
Mensajes de error e información para ayudarle con los problemas comunes de DB2.

Ejemplos
Enlaces a listados HTML de los programas de ejemplo proporcionados con DB2.

Guías de aprendizaje
Ayuda con instrucciones diseñada para ayudarle a conocer una característica de DB2.

Prerrequisitos:

Algunos enlaces del Centro de información de DB2 apuntan a sitios Web de Internet. Para visualizar el contenido de estos enlaces, primero tendrá que conectarse a Internet.

Procedimiento:

Para buscar información del producto accediendo al Centro de información de DB2 desde las herramientas:

1. Inicie el Centro de información de DB2, de uno de los modos siguientes:
   - Desde las herramientas de administración gráficas, pulse el icono **Centro de información** de la barra de herramientas. También lo puede seleccionar desde el menú **Ayuda**.
   - En la línea de mandatos, entre `db2ic`.
2. Pulse la pestaña del tipo de información relacionado con la información que está buscando.
3. Navegue por el árbol y pulse el tema en el que está interesado. El Centro de información lanzará un navegador Web para visualizar la información.
4. Para buscar información sin examinar las listas, pulse el icono **Buscar** situado a la derecha de la lista.

   Una vez que el Centro de información haya lanzado un navegador para visualizar la información, podrá realizar una búsqueda de texto completo pulsando el icono **Buscar** en la barra de herramientas de navegación.

Conceptos relacionados:

- "Accesibilidad" en la página 595
- "Acceso al Centro de información de DB2 desde un navegador" en la página 597

Tareas relacionadas:

- “Búsqueda de temas mediante el acceso al Centro de información de DB2 desde un navegador” en la página 585
- “Búsqueda en la documentación de DB2” en la página 593

Cómo ver documentación técnica en línea directamente desde el CD de documentación HTML de DB2

Todos los temas HTML que se pueden instalar desde el *CD de documentación HTML de DB2* también se pueden leer directamente del CD. Por consiguiente, puede ver la documentación sin tener que instalarla.

Restricciones:

Dado que la ayuda de Herramientas se instala desde el CD del producto DB2 y no desde el *CD de documentación HTML de DB2*, deberá instalar el producto DB2 para poder ver la ayuda.

Procedimiento:

1. Inserte el *CD de documentación HTML de DB2*. En los sistemas operativos UNIX, monte el *CD de documentación HTML de DB2*. Consulte el manual *Iniciación rápida* para obtener información más detallada sobre cómo montar un CD en sistemas operativos UNIX.
2. Inicie el navegador HTML y abra el archivo apropiado:
   - Para sistemas operativos Windows:

     ```
     e:\archivos de programa\IBM\SQLLIB\doc\htmlcd\%L\index.htm
     ```

     donde *e* representa la unidad de CD-ROM y %L es el entorno nacional de la documentación que desea utilizar, por ejemplo es_ES para el español.
   - Para sistemas operativos UNIX:

     ```
     /cdrom/archivos de programa/IBM/SQLLIB/doc/htmlcd/%L/index.htm
     ```

     donde */cdrom/* representa el lugar en el que está montado el CD y %L es el entorno nacional de la documentación que desea utilizar, por ejemplo es_Es para el español.

Tareas relacionadas:

- “Búsqueda de temas mediante el acceso al Centro de información de DB2 desde un navegador” en la página 585
- “Copia de archivos desde el CD de documentación HTML de DB2 en un servidor Web” en la página 591

Consulta relacionada:

• “Visión general de la información técnica de DB2 Universal Database” en la página 573

Actualización de la documentación HTML instalada en la máquina

Ahora es posible actualizar el código HTML instalado desde el *CD de documentación HTML de DB2* cuando IBM pone las actualizaciones a disposición de los usuarios. Esta tarea puede realizarse de una de las dos maneras siguientes:

- Utilizando el Centro de información (si tiene instaladas las herramientas de GUI de administración de DB2).
- Bajando y aplicando un FixPak de documentación HTML de DB2.

Nota: Esto NO actualizará el código de DB2; sólo actualizará la documentación HTML instalada desde el *CD de documentación HTML de DB2*.

Procedimiento:

Si desea utilizar el Centro de información para actualizar la documentación local:

1. Inicie el Centro de información de DB2, de uno de los modos siguientes:
   - Desde las herramientas de administración gráficas, pulse el icono **Centro de información** de la barra de herramientas. También lo puede seleccionar desde el menú **Ayuda**.
   - En la línea de mandatos, entre `db2ic`.
2. Asegúrese de que la máquina tiene acceso a la Internet externa; el actualizador bajará el FixPak de documentación más reciente del servidor IBM si es necesario.
3. Seleccione **Centro de información** —> **Actualizar documentación local** en el menú para iniciar la actualización.
4. Proporcione la información de proxy (si es necesaria) para conectarse a la Internet externa.

El Centro de información bajará y aplicará el FixPak de documentación más reciente, si hay alguno disponible.

Para bajar y aplicar manualmente el FixPak de documentación:

1. Asegúrese de que la máquina está conectada a Internet.
2. Abra la página de soporte de DB2 en el navegador Web, en la siguiente dirección: www.ibm.com/software/data/db2/udb/winos2unix/support.
3. Siga el enlace correspondiente a la Versión 8 y busque el enlace ″FixPaks de documentación″.

4. Determine si la versión de la documentación local está anticuada, comparando el nivel de FixPak de documentación con el nivel de documentación que tiene instalado. Esta documentación actual de la máquina está en el nivel siguiente: **DB2 v8.1 GA**.
5. Si se encuentra disponible una versión más reciente de la documentación, baje el FixPak aplicable al sistema operativo. Existe un FixPak para todas las plataformas Windows y un FixPak para todas las plataformas UNIX.
6. Aplique el FixPak:
   - Para sistemas operativos Windows: el FixPak de documentación es un archivo zip de autoextracción. Coloque el FixPak de documentación que ha bajado en un directorio vacío y ejecútelo. Creará un mandato `setup` que puede ejecutar para instalar el FixPak de documentación.
   - Para sistemas operativos UNIX: el FixPak de documentación es un archivo tar.Z comprimido. Descomprima y desempaquete el archivo mediante tar. Creará un directorio denominado `delta_install` con un script denominado `installdocfix`. Ejecute este script para instalar el FixPak de documentación.

Tareas relacionadas:

- "Copia de archivos desde el CD de documentación HTML de DB2 en un servidor Web" en la página 591

Consulta relacionada:

- "Visión general de la información técnica de DB2 Universal Database" en la página 573

Copia de archivos desde el CD de documentación HTML de DB2 en un servidor Web

La biblioteca de información de DB2 entera se entrega al usuario en el *CD de documentación HTML de DB2* y se puede instalar en un servidor Web para acceder más fácilmente a ella. Simplemente copie en el servidor Web la documentación para los idiomas que desee.

Nota: Puede encontrarse con un rendimiento bajo si accede a la documentación HTML desde un servidor Web mediante una conexión de baja velocidad.

Procedimiento:

Para copiar archivos desde el *CD de documentación HTML de DB2* en un servidor Web, utilice la vía de acceso origen apropiada:

- Para sistemas operativos Windows:

```
E:\archivos de programa\IBM\SQLLIB\doc\htmlcd\%L\*.*
```

donde *E* representa la unidad de CD-ROM y *%L* representa el identificador de idioma.

- Para sistemas operativos UNIX:

  /*cdrom*/archivos de programa/IBM/SQLLIB/doc/htmlcd/*%L*/*.*

  donde *cdrom* representa el punto de montaje para la unidad de CD-ROM y *%L* representa el identificador de idioma

Tareas relacionadas:

- "Búsqueda en la documentación de DB2" en la página 593

Consulta relacionada:

- "Idiomas, entornos locales y páginas de códigos soportados por DB2" en el manual *Guía rápida de iniciación para servidores de DB2*
- "Visión general de la información técnica de DB2 Universal Database" en la página 573

Resolución de problemas de búsqueda de documentación de DB2 con Netscape 4.x

La mayoría de los problemas de búsqueda están relacionados con el soporte Java proporcionado por los navegadores web. Esta tarea describe soluciones provisionales posibles.

Procedimiento:

Un problema común con Netscape 4.x implica la ausencia o la colocación errónea de una clase de seguridad. Intente la solución provisional siguiente, especialmente si ve la línea siguiente en la consola Java del navegador:

```
Cannot find class  java/security/InvalidParameterException
```

- En sistemas operativos Windows:

  Desde el *CD de documentación HTML de DB2*, copie el archivo *x*:archivos de programa\IBM\SQLLIB\doc\htmlcd*entorno_nacional*\InvalidParameterException.class proporcionado en el directorio java\classes\java\security\ relativo a la instalación del navegador Netscape, donde *x* representa la letra de unidad de CD-ROM y *entorno_nacional* representa el nombre del entorno nacional deseado.

  Nota: Puede que tenga que crear la estructura de subdirectorios java\security\.

- En sistemas operativos UNIX:

  Desde el *CD de documentación HTML de DB2*, copie el archivo /*cdrom*/archivos de programa/IBM/SQLLIB/doc/htmlcd/*entorno_nacional*

/InvalidParameterException.class proporcionado en el directorio java/classes/java/security/ relativo a la instalación del navegador Netscape, donde *cdrom* representa el punto de montaje del CD-ROM y *entorno_nacional* representa el nombre del entorno nacional deseado.

Nota: Puede que tenga que crear la estructura de subdirectorios java/security/.

Si el navegador Netscape sigue sin visualizar la ventana de entrada de búsqueda, intente lo siguiente:

- Detenga todas las instancias de los navegadores Netscape para asegurarse de que no hay ningún código de Netscape en ejecución en la máquina. A continuación, abra una instancia nueva del navegador Netscape e intente iniciar la búsqueda otra vez.
- Depure la antememoria del navegador.
- Pruebe una versión diferente de Netscape o un navegador diferente.

Tareas relacionadas:

- "Búsqueda en la documentación de DB2" en la página 593

Búsqueda en la documentación de DB2

Puede efectuar búsquedas en la biblioteca de documentación de DB2 para encontrar la información que necesita. Al pulsar el icono de búsqueda de la barra de herramientas de navegación del Centro de información de DB2 (al que se accede desde un navegador), se abre una ventana emergente. La búsqueda pueda tardar un minuto en cargarse, en función de la velocidad del sistema y de la red.

Prerrequisitos:

Necesita Netscape 6.1 o posterior, o bien Internet Explorer 5 o posterior de Microsoft. Asegúrese de que el soporte Java del navegador esté habilitado.

Restricciones:

Al utilizar la búsqueda de documentación se aplican las restricciones siguientes:

- La búsqueda no distingue entre mayúsculas y minúsculas.
- No se soportan las búsquedas booleanas.
- No se da soporte a búsquedas mediante caracteres comodín o parciales. Una búsqueda de *java** (o *java*) sólo buscará la serie literal *java** (o *java*) y no encontrará, por ejemplo, *javadoc*.

Procedimiento:

Si desea realizar búsquedas en la documentación de DB2:

1. En la barra de herramientas de navegación, pulse el icono **Buscar**.
2. En el campo de entrada de texto superior de la ventana Buscar, escriba uno o varios términos (separados por un espacio) relacionados con el área de interés y pulse **Buscar**. Se visualiza una lista de temas clasificados por exactitud en el campo **Resultados**. La clasificación numérica junto al acierto proporciona una indicación de la importancia de la coincidencia (los números más altos indican resultados coincidentes más importantes).

   La entrada de más términos aumentará la precisión de la consulta al mismo tiempo que reducirá el número de temas devueltos.
3. En la lista **Resultados**, pulse el título del tema que desea leer. El tema se visualiza en el marco de contenido del Centro de información de DB2.

Nota: Cuando se realiza una búsqueda, el primer resultado (clasificación más alta) se carga automáticamente en el marco del navegador. Para ver el contenido de otros resultados de la búsqueda, pulse el resultado en la lista de resultados.

Tareas relacionadas:

- "Resolución de problemas de búsqueda de documentación de DB2 con Netscape 4.x" en la página 592

Información en línea de resolución de problemas de DB2

En el release de DB2 UDB Versión 8, ya no hay una *Guía para la resolución de problemas*. La información de resolución de problemas que antes estaba contenida en esta guía se ha integrado en las publicaciones de DB2. De este modo, podemos proporcionar la información más actualizada posible. Para buscar información sobre los programas de utilidad y las funciones para la resolución de problemas de DB2, acceda el Centro de información de DB2 desde cualquiera de las herramientas.

Consulte el sitio de soporte en línea de DB2 si tiene problemas y desea obtener ayuda para encontrar las causas y las soluciones posibles. El sitio de soporte contiene una gran base de datos de publicaciones, notas técnicas, registros APAR (problema de producto), FixPaks y otros recursos de DB2 que se actualiza constantemente. Puede utilizar el sitio de soporte para buscar en esta base de conocimiento y encontrar posibles soluciones a los problemas.

Acceda al sitio de Soporte en línea en la dirección www.ibm.com/software/data/db2/udb/winos2unix/support o pulsando el botón **Soporte en línea** en el Centro de información de DB2. En este sitio,

ahora también se encuentra disponible información que cambia frecuentemente, por ejemplo el listado de códigos de error internos de DB2.

Conceptos relacionados:

- "Acceso al Centro de información de DB2 desde un navegador" en la página 597

Tareas relacionadas:

- "Búsqueda de información de productos mediante el acceso al Centro de información de DB2 desde las herramientas de administración" en la página 587

Accesibilidad

Las características de accesibilidad ayudan a los usuarios con discapacidades físicas, por ejemplo movilidad o visión limitada, a utilizar los productos de software satisfactoriamente. En DB2 Universal Database Versión 8, las características de accesibilidad principales son las siguientes:

- DB2 permite trabajar con todas las características utilizando el teclado en lugar del ratón. Consulte "Entrada de teclado y navegación".
- DB2 permite personalizar el tamaño y el color de los fonts. Consulte "Pantalla accesible" en la página 596.
- DB2 permite recibir señales de alerta visuales o sonoras. Consulte "Señales de alerta alternativas" en la página 596.
- DB2 soporta las aplicaciones de accesibilidad que utilizan la API de accesibilidad de Java. Consulte "Compatibilidad con tecnologías de asistencia" en la página 596.
- DB2 viene con documentación que se proporciona en un formato accesible. Consulte "Documentación accesible" en la página 596.

Entrada de teclado y navegación

Entrada de teclado

Puede trabajar con las Herramientas de DB2 utilizando sólo el teclado. Puede utilizar teclas o combinaciones de teclas para llevar a cabo la mayoría de las operaciones que también se pueden realizar con el ratón.

Foco del teclado

En sistemas basados en UNIX, la posición del foco del teclado está resaltada, lo que indica qué área de la ventana está activa y dónde serán efectivas las pulsaciones.

Pantalla accesible

Las Herramientas de DB2 tienen características que amplían la interfaz de usuario y mejoran la accesibilidad para los usuarios con visión reducida. Estas mejoras de la accesibilidad incluyen soporte para propiedades de font personalizables.

Valores de font

Las Herramientas de DB2 permiten seleccionar el color, el tamaño y el font para el texto de los menús y las ventanas de diálogo, utilizando el cuaderno Valores de herramientas.

No dependencia del color

No es necesario distinguir los colores para utilizar cualquiera de las funciones de este producto.

Señales de alerta alternativas

Puede especificar si desea recibir las alertas a través de señales visuales o sonoras, utilizando el cuaderno Valores de herramientas.

Compatibilidad con tecnologías de asistencia

La interfaz de las Herramientas de DB2 soporta la API de accesibilidad de Java que permite el uso de lectores de pantalla y otras tecnologías de asistencia utilizadas por las personas discapacitadas.

Documentación accesible

La documentación para la familia de productos DB2 está disponible en formato HTML. Esto permite ver la documentación de acuerdo con las preferencias de pantalla establecidas en el navegador. También permite utilizar lectores de pantalla y otras tecnologías de asistencia.

Guías de aprendizaje de DB2

Las guías de aprendizaje de DB2 ayudan a conocer los diversos aspectos de DB2 Universal Database. Las guías de aprendizaje proporcionan ejercicios con instrucciones paso a paso en las áreas de desarrollo de aplicaciones, ajuste del rendimiento de las consultas de SQL, trabajo con depósitos de datos, gestión de metadatos y desarrollo de servicios Web utilizando DB2.

Antes de empezar:

Para poder acceder a estas guías de aprendizaje utilizando los enlaces que figuran abajo, deberá instalar las guías de aprendizaje desde el *CD de documentación HTML de DB2*.

Si no desea instalar las guías de aprendizaje, puede ver las versiones HTML de las mismas directamente desde el *CD de documentación HTML de DB2*. En el *CD de documentación PDF de DB2* también se encuentran disponibles versiones PDF de estas guías de aprendizaje.

Algunos ejercicios de las guías de aprendizaje utilizan datos o código de ejemplo. Consulte cada guía de aprendizaje individual para obtener una descripción de los prerrequisitos para las tareas específicas.

Guías de aprendizaje de DB2 Universal Database:

Si ha instalado las guías de aprendizaje desde el *CD de documentación HTML de DB2*, puede pulsar el título de una guía de aprendizaje de la lista que se indica a continuación para ver dicha guía.

Guía de aprendizaje de Inteligencia empresarial: Introducción al Centro de depósito de datos
Realizar tareas de introducción de depósito de datos utilizando el Centro de depósito de datos.

Guía de aprendizaje de Inteligencia empresarial: Lecciones ampliadas sobre el depósito de datos
Realizar tareas avanzadas de depósito de datos utilizando el Centro de depósito de datos.

Guía de aprendizaje del Centro de desarrollo para Video Online utilizando Microsoft Visual Basic
Crear diversos componentes de una aplicación utilizando la Macro automática del Centro de desarrollo para Microsoft Visual Basic.

Guía de aprendizaje del Centro de catálogos de información
Crear y gestionar un catálogo de información para localizar y usar metadatos utilizando el Centro de catálogos de información.

Guía de aprendizaje de Video Central para e-business
Desarrollar y desplegar una aplicación avanzada de Servicios Web DB2 utilizando productos WebSphere.

Guía de aprendizaje de Visual Explain
Analizar, optimizar y ajustar sentencias de SQL para obtener un mejor rendimiento al utilizar Visual Explain.

Acceso al Centro de información de DB2 desde un navegador

El Centro de información de DB2 proporciona acceso a toda la información que necesita para obtener el máximo provecho de DB2 Universal Database y DB2 Connect en su empresa. El Centro de información de DB2 también documenta las características y los componentes principales de DB2,

incluyendo la duplicación, el depósito de datos, el Centro de catálogos de información, Life Sciences Data Connect y DB2 Extenders.

El Centro de información de DB2 al que se accede desde un navegador presenta las características siguientes si se visualiza en Netscape Navigator 6.1 o posterior o bien Microsoft Internet Explorer 5 o posterior. Algunas características requieren que se habilite el soporte de Java o JavaScript:

Documentación actualizada con regularidad
Mantenga los temas actualizados bajando el código HTML actualizado.

Búsqueda
Busque todos los temas instalados en la estación de trabajo pulsando **Buscar** en la barra de herramientas de navegación.

Árbol de navegación integrado
Localice cualquier tema de la biblioteca de DB2 desde un solo árbol de navegación. El árbol de navegación está organizado por tipo de información, como se indica a continuación:

- Las tareas proporcionan instrucciones paso a paso sobre cómo lograr un objetivo.
- Los conceptos proporcionan una visión general de un tema.
- Los temas de consulta proporcionan información detallada sobre un tema, incluyendo la sintaxis de sentencias y mandatos, la ayuda de mensajes y los requisitos.

Índice maestro
Acceda a la información instalada desde el *CD de documentación HTML de DB2* desde el índice maestro. El índice está organizado en orden alfabético por términos del índice.

Glosario maestro
El glosario maestro define los términos utilizados en el Centro de información de DB2. El glosario está organizado en orden alfabético por términos del glosario.

Tareas relacionadas:

- "Búsqueda de temas mediante el acceso al Centro de información de DB2 desde un navegador" en la página 585
- "Búsqueda de información de productos mediante el acceso al Centro de información de DB2 desde las herramientas de administración" en la página 587
- "Actualización de la documentación HTML instalada en la máquina" en la página 590

Avisos

Es posible que IBM no comercialice en todos los países algunos productos, servicios o características descritos en este manual. Consulte al representante local de IBM para obtener información sobre los productos y servicios que actualmente pueden adquirirse en su zona. Cualquier referencia a un producto, programa o servicio de IBM no pretende afirmar ni implicar que sólo se pueda utilizar dicho producto, programa o servicio de IBM. En su lugar se puede utilizar cualquier producto, programa o servicio funcionalmente equivalente que no vulnere ninguno de los derechos de propiedad intelectual de IBM. Sin embargo, es responsabilidad del usuario evaluar y verificar el funcionamiento de cualquier producto, programa o servicio que no sea de IBM.

IBM puede tener patentes o solicitudes de patentes en tramitación que afecten al tema tratado en este documento. La posesión de este documento no confiere ninguna licencia sobre dichas patentes. Puede realizar consultas sobre licencias escribiendo a:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
EE.UU.

Para realizar consultas sobre licencias referentes a información de doble byte (DBCS), puede ponerse en contacto con el Departamento de Propiedad Intelectual de IBM de su país/región o escribir a:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokio 106, Japón

El párrafo siguiente no es aplicable al Reino Unido ni a ningún país/región en donde tales disposiciones sean incompatibles con la legislación local: INTERNATIONAL BUSINESS MACHINES CORPORATION PROPORCIONA ESTA PUBLICACIÓN "TAL CUAL", SIN GARANTÍA DE NINGUNA CLASE, NI EXPLÍCITA NI IMPLÍCITA, INCLUIDAS, PERO SIN LIMITARSE A ELLAS, LAS GARANTÍAS IMPLÍCITAS DE NO VULNERACIÓN DE DERECHOS, COMERCIALIZACIÓN O IDONEIDAD PARA UN FIN DETERMINADO. Algunos estados no permiten la exclusión de garantías expresas o implícitas en determinadas transacciones, por lo que es posible que esta declaración no sea aplicable en su caso.

Esta publicación puede contener inexactitudes técnicas o errores tipográficos. Periódicamente se efectúan cambios en la información aquí contenida; dichos cambios se incorporarán a las nuevas ediciones de la publicación. IBM puede efectuar, en cualquier momento y sin previo aviso, mejoras y cambios en los productos y programas descritos en esta publicación.

Las referencias hechas en esta publicación a sitios Web que no son de IBM se proporcionan sólo para la comodidad del usuario y no constituyen un aval de esos sitios Web. La información contenida en esos sitios Web no forma parte de la información del presente producto IBM y el usuario es responsable de la utilización de dichos sitios Web.

IBM puede utilizar o distribuir cualquier información que se le facilite de la manera que considere adecuada, sin contraer por ello ninguna obligación con el remitente.

Los licenciatarios de este programa que deseen obtener información sobre él con el fin de habilitar: (i) el intercambio de información entre programas creados de forma independiente y otros programas (incluido éste) y (ii) el uso mutuo de la información intercambiada, deben ponerse en contacto con:

IBM Canada Limited
Office of the Lab Director
8200 Warden Avenue
Markham, Ontario
L6G 1C7
CANADÁ

Dicha información puede estar disponible, sujeta a los términos y condiciones apropiados, incluido en algunos casos el pago de una tarifa.

El programa bajo licencia descrito en este documento y todo el material bajo licencia asociado a él, los proporciona IBM según los términos del Acuerdo de Cliente de IBM, el Acuerdo Internacional de Programas Bajo Licencia de IBM o cualquier acuerdo equivalente entre el usuario e IBM.

Los datos de rendimiento contenidos en este documento se obtuvieron en un entorno controlado. Por lo tanto, los resultados obtenidos en otros entornos operativos pueden variar significativamente. Algunas mediciones pueden haberse realizado en sistemas experimentales y no es seguro que estas mediciones sean las mismas en los sistemas disponibles comercialmente. Además, algunas mediciones pueden haberse calculado mediante extrapolación. Los resultados reales pueden variar. Los usuarios del presente manual deben verificar los datos aplicables para su entorno específico.

La información referente a productos que no son de IBM se ha obtenido de los proveedores de esos productos, de sus anuncios publicados o de otras

fuentes disponibles públicamente. IBM no ha probado esos productos y no puede confirmar la exactitud del rendimiento, la compatibilidad ni ninguna otra afirmación referente a productos que no son de IBM. Las preguntas sobre las prestaciones de productos que no son de IBM deben dirigirse a los proveedores de esos productos.

Todas las declaraciones de intenciones de IBM están sujetas a cambio o cancelación sin previo aviso, y sólo representan objetivos.

Este manual puede contener ejemplos de datos e informes que se utilizan en operaciones comerciales diarias. Para ilustrarlos de la forma más completa posible, los ejemplos incluyen nombres de personas, empresas, marcas y productos. Todos estos nombres son ficticios y cualquier similitud con nombres y direcciones utilizados por una empresa real es totalmente fortuita.

LICENCIA DE COPYRIGHT:

Este manual puede contener programas de aplicaciones de ejemplo escritos en lenguaje fuente, que muestran técnicas de programación en diversas plataformas operativas. Puede copiar, modificar y distribuir estos programas de ejemplo como desee, sin pago alguno a IBM, con la intención de desarrollar, utilizar, comercializar o distribuir programas de aplicaciones de acuerdo con la interfaz de programación de aplicaciones correspondiente a la plataforma operativa para la que están escritos los programas de ejemplo. Estos ejemplos no se han probado exhaustivamente bajo todas las condiciones. Por lo tanto, IBM no puede asegurar ni implicar la fiabilidad, utilidad o función de estos programas.

Cada copia o parte de estos programas de ejemplo o cualquier trabajo derivado debe incluir una nota de copyright como la siguiente:

© (*nombre de la empresa*) (*año*). Partes de este código proceden de programas de ejemplo de IBM Corp. © Copyright IBM Corp. *_entre el o los años_*. Reservados todos los derechos.

Marcas registradas

Los términos siguientes son marcas registradas de International Business Machines Corporation en los EE.UU. y/o en otros países y se han utilizado como mínimo en uno de los documentos de la biblioteca de documentación de DB2 UDB.

ACF/VTAM
AISPO
AIX
AIXwindows
AnyNet
APPN
AS/400
BookManager
C Set++
C/370
CICS
Database 2
DataHub
DataJoiner
DataPropagator
DataRefresher
DB2
DB2 Connect
DB2 Extenders
DB2 OLAP Server
DB2 Universal Database
Distributed Relational Database Architecture
DRDA
eServer
Extended Services
FFST
First Failure Support Technology
IBM
IMS
IMS/ESA
iSeries
LAN Distance
MVS
MVS/ESA
MVS/XA
Net.Data
NetView
OS/390
OS/400
PowerPC
pSeries
QBIC
QMF
RACF
RISC System/6000
RS/6000
S/370
SP
SQL/400
SQL/DS
System/370
System/390
SystemView
Tivoli
VisualAge
VM/ESA
VSE/ESA
VTAM
WebExplorer
WebSphere
WIN-OS/2
z/OS
zSeries

Los términos siguientes son marcas registradas de otras empresas y se han utilizado como mínimo en uno de los documentos de la biblioteca de documentación de DB2 UDB:

Microsoft, Windows, Windows NT y el logotipo de Windows son marcas registradas de Microsoft Corporation en los EE.UU. y/o en otros países.

Intel y Pentium son marcas registradas de Intel Corporation en los EE.UU. y/o en otros países.

Java y todas las marcas registradas basadas en Java son marcas registradas de Sun Microsystems, Inc. en los EE.UU. y/o en otros países.

UNIX es marca registrada de The Open Group en los EE.UU. y/o en otros países.

Otros nombres de empresas, productos o servicios, pueden ser marcas registradas o marcas de servicio de otras empresas.

Cómo ponerse en contacto con IBM

En los EE.UU., puede ponerse en contacto con IBM llamando a uno de los siguientes números:

- 1-800-237-5511 para servicio al cliente
- 1-888-426-4343 para obtener información sobre las opciones de servicio técnico disponibles
- 1-800-IBM-4YOU (426-4968) para márketing y ventas de DB2

En Canadá, puede ponerse en contacto con IBM llamando a uno de los siguientes números:

- 1-800-IBM-SERV (1-800-426-7378) para servicio al cliente
- 1-800-465-9600 para obtener información sobre las opciones de servicio técnico disponibles
- 1-800-IBM-4YOU (1-800-426-4968) para márketing y ventas de DB2

Para localizar una oficina de IBM en su país o región, consulte IBM Directory of Worldwide Contacts en el sitio Web www.ibm.com/planetwide

Información sobre productos

La información relacionada con productos DB2 Universal Database se encuentra disponible por teléfono o a través de la World Wide Web en el sitio www.ibm.com/software/data/db2/udb

Este sitio contiene la información más reciente sobre la biblioteca técnica, pedidos de manuales, descargas de clientes, grupos de noticias, FixPacks, novedades y enlaces con recursos de la Web.

Si vive en los EE.UU., puede llamar a uno de los números siguientes:

- 1-800-IBM-CALL (1-800-426-2255) para solicitar productos u obtener información general.
- 1-800-879-2755 para solicitar publicaciones.

Para obtener información sobre cómo ponerse en contacto con IBM desde fuera de los EE.UU., vaya a la página IBM Worldwide en el sitio www.ibm.com/planetwide

Número Pieza: CT17TES

Printed in Denmark by IBM Danmark A/S

(1P) P/N: CT17TES

SC10-3723-00