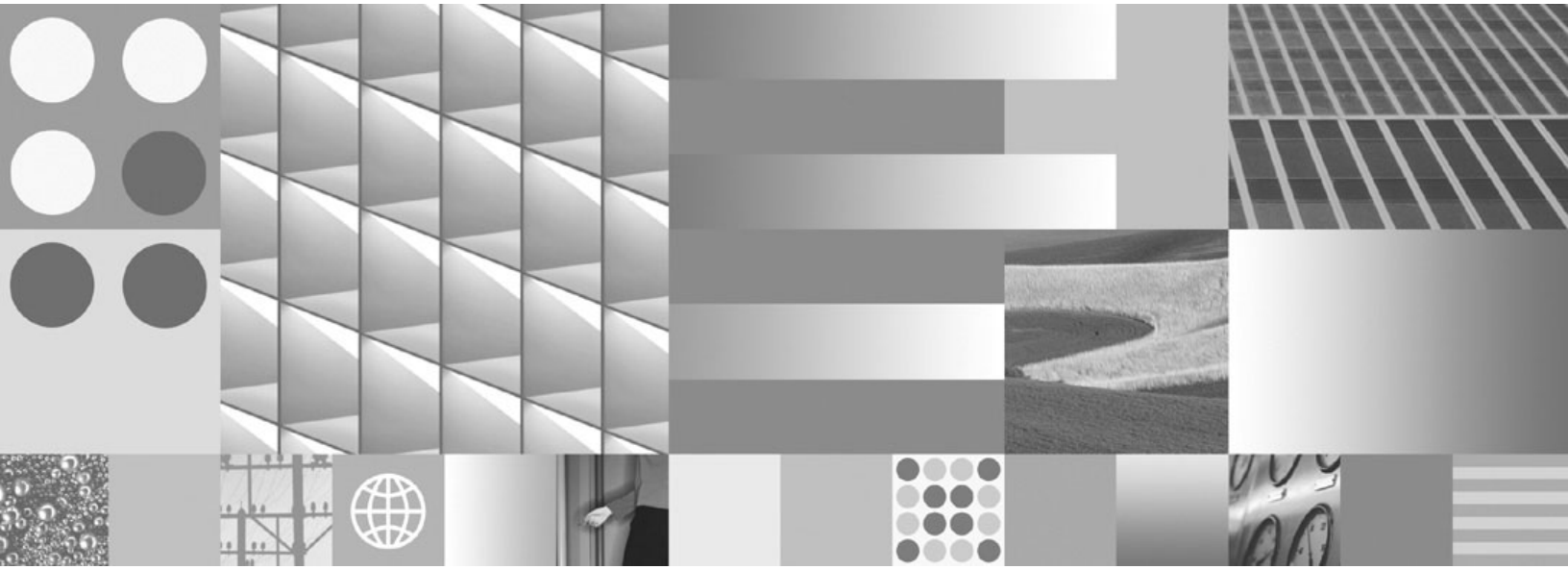
DB2 Versión 9.5
para Linux, UNIX y Windows

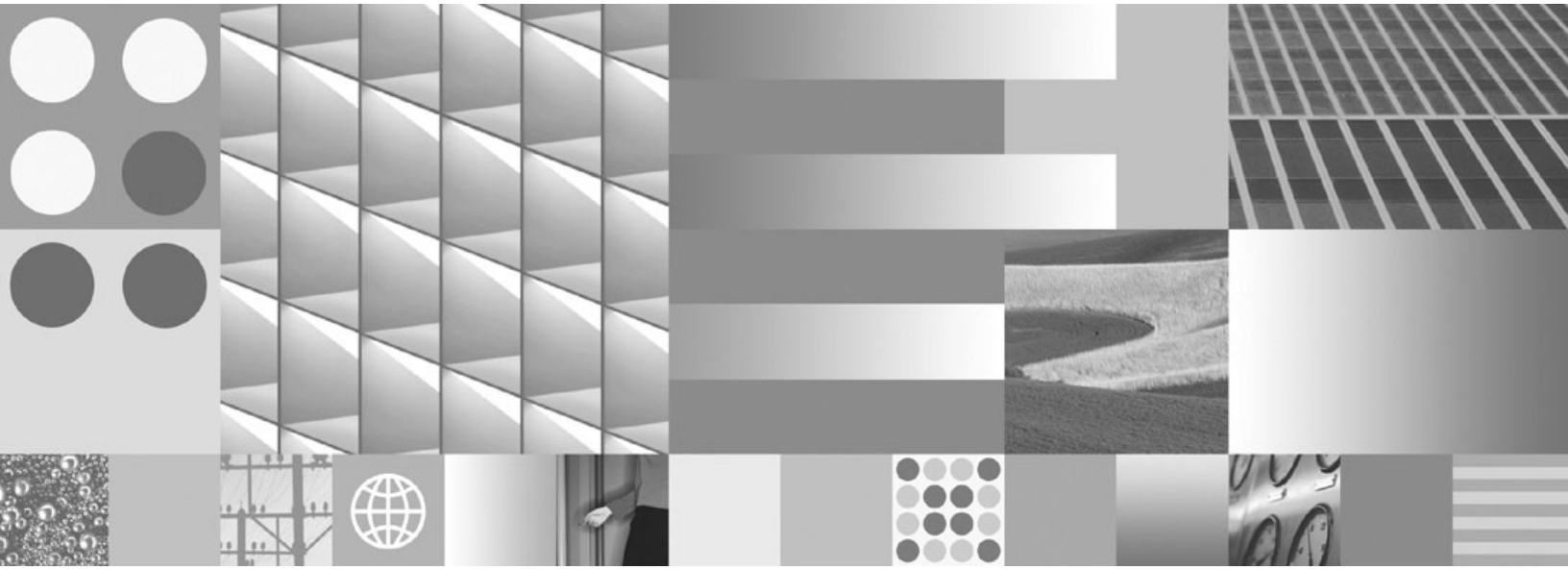


Desarrollo de aplicaciones de SQL incorporado

Actualizado en marzo de 2008

SC11-3500-01

DB2 Versión 9.5
para Linux, UNIX y Windows



Desarrollo de aplicaciones de SQL incorporado

Actualizado en marzo de 2008

SC11-3500-01

Nota

Antes de utilizar esta información y el producto al que da soporte, lea la información general contenida en el apartado Apéndice B, "Avisos", en la página 223.

Nota de edición

Esta publicación es la traducción del original inglés: DB2 Version 9.5 for Linux, UNIX, and Windows - Developing Embedded SQL Applications, (SC23-5852-01).

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

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

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

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

Contenido

Parte 1. Información preliminar sobre el SQL incorporado

Las aplicaciones de bases de datos de SQL incorporado se conectan a bases de datos y ejecutan sentencias de SQL incorporado. Las sentencias de SQL incorporado se incorporan dentro de una aplicación de lenguaje principal. Las aplicaciones de bases de datos de SQL incorporado dan soporte a la incorporación de sentencias de SQL que se tienen que ejecutar estática o dinámicamente.

Puede desarrollar aplicaciones de SQL incorporado para DB2 en los siguientes lenguajes de programación principales: C, C++, COBOL, FORTRAN y REXX.

Nota: El soporte de SQL incorporado en FORTRAN y REXX ha quedado obsoleto y permanecerá a nivel de DB2 Universal Database , Versión 5.2.

La creación de aplicaciones de SQL incorporado incluye dos pasos previos a la compilación y enlace de la aplicación.

- Preparación de los archivos fuentes que contienen sentencias de SQL incorporado mediante el precompilador de DB2.

  El mandato `PREP (PRECOMPILE)` sirve para invocar el precompilador de DB2, que lee el código fuente, lo analiza y convierte las sentencias de SQL incorporado en llamadas a las API de servicios de tiempo de ejecución de DB2, y finalmente escribe la salida en un nuevo archivo fuente modificado. El precompilador genera planes de acceso para las sentencias de SQL que se almacenan juntas como un paquete dentro de la base de datos.
- Vinculación de las sentencias de la aplicación con la base de datos de destino.

  La vinculación se realiza por omisión durante la precompilación (el mandato `PREP`). Si la vinculación se tiene que diferir (por ejemplo, el mandato `BIND` se tiene que ejecutar más tarde), se tiene que especificar la opción `BINDFILE` en el momento de especificar `PREP` para que se genere un archivo de vinculación.

Cuando haya precompilado y vinculado la aplicación de SQL incorporado, ya está lista para que se compile y se enlace mediante las herramientas de desarrollo específicas del lenguaje principal.

Para ayudar en el desarrollo de aplicaciones de SQL incorporado, puede consultar la plantilla de SQL incorporado en C. En el directorio `%DB2PATH%\SQLLIB\samples` también se pueden encontrar ejemplos de aplicaciones de ejemplo de SQL incorporado que funcionan.

Nota: %DB2PATH% es el directorio de instalación de DB2

SQL estático y dinámico

Las sentencias de SQL se pueden ejecutar de dos formas: estática y dinámicamente.

Sentencias de SQL ejecutadas estáticamente

Para las sentencias de SQL ejecutadas estáticamente, la sintaxis se conoce por completo en el momento de la precompilación. 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 por ejemplo, tipos de datos, debe estar ya precompilada. Debe precompilar, vincular y compilar las sentencias de SQL ejecutadas estáticamente antes de ejecutar la aplicación. El SQL estático se utiliza mejor en bases de datos cuyas estadísticas no cambian mucho.

Sentencias de SQL ejecutadas dinámicamente
La aplicación crea y ejecuta las sentencias de SQL ejecutadas dinámicamente en el momento de la 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 una situación adecuada para SQL dinámico.

Capítulo 1. Incorporación de sentencias de SQL en un lenguaje principal

Structured Query Language (SQL) es un lenguaje estandarizado que se puede utilizar para manipular objetos de bases de datos y los datos que contienen. A pesar de las diferencias entre los lenguajes principales, todas las aplicaciones de SQL incorporado están formadas por tres elementos necesarios para configurar y ejecutar una sentencia de SQL:

1. Una DECLARE SECTION para declarar variables del lenguaje principal. No es necesario que la declaración de la estructura SQLCA esté en la sección DECLARE.
2. El cuerpo principal de la aplicación, que consta de la configuración y ejecución de sentencias de SQL.
3. Colocaciones de lógica que confirma o retrotrae los cambios realizados por las sentencias de SQL.

Para cada lenguaje principal, hay ciertas diferencias entre las directrices generales que se aplican a todos los lenguajes y las reglas específicas de los lenguajes individuales.

Sentencias de SQL incorporado en aplicaciones C y C++

Las aplicaciones C y C++ de SQL incorporado constan de tres elementos principales para configurar y ejecutar una sentencia de SQL.

- Una DECLARE SECTION para declarar variables del lenguaje principal. No es necesario que la declaración de la estructura SQLCA esté en la sección DECLARE.
- El cuerpo principal de la aplicación, que consta de la configuración y ejecución de sentencias de SQL
- Colocaciones de lógica que confirman o retrotraen los cambios realizados por las sentencias de SQL

Sintaxis correcta de elementos de C y C++

Inicializador de la sentencia
EXEC SQL

Serie de la sentencia
Cualquier sentencia de SQL válida

Terminador de la sentencia
Punto y coma (;)

Por ejemplo, para ejecutar una sentencia de SQL estáticamente dentro de una aplicación C, podría incluir lo siguiente dentro del código de la aplicación:

```
EXEC SQL SELECT col INTO :hostvar FROM table;
```

El ejemplo siguiente muestra cómo ejecutar una sentencia de SQL dinámicamente utilizando la variable del lenguaje principal `stmt1`:

```
strcpy(stmt1, "CREATE TABLE table1(col1 INTEGER)");
EXEC SQL EXECUTE IMMEDIATE :stmt1;
```

Las siguientes directrices y reglas se aplican a la ejecución de sentencias de SQL incorporado en aplicaciones C y C++:

- Puede empezar la serie de la sentencia de SQL en la misma línea que el inicializador de sentencia EXEC SQL.
- No divida 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 y C++ delante del inicializador de la sentencia o detrás del terminador de la sentencia.
- Se pueden poner varias sentencias de SQL y de C o C++ en la misma línea. Por ejemplo:

  ```
  EXEC SQL OPEN c1; if (SQLCODE >= 0) EXEC SQL FETCH c1 INTO :hv;
  ```

- Los retornos de carro, saltos de línea y tabulaciones se pueden incluir dentro de una serie entrecomillada. El precompilador de SQL los dejará tal cual.
- No utilice la sentencia #include 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 #include. En su lugar, utilice la sentencia INCLUDE de SQL para importar los archivos de inclusión.
- Están permitidos los comentarios de SQL en cualquier línea que forme parte de una sentencia de SQL incorporado, con a excepción de las sentencias ejecutadas dinámicamente.
  - 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. Estos comentarios de SQL provocan errores de compilación ya que los compiladores los interpretan como sintaxis de C o C++.
  - Puede utilizar comentarios de SQL en una serie de una sentencia estática siempre que se permitan caracteres en blanco.
  - El uso de los delimitadores de comentarios de C y C++ `/* */` está permitido en las sentencias de SQL incorporado tanto estáticas como dinámicas.
  - La utilización de comentarios de C++ de estilo `//` no están permitidos dentro de sentencias de SQL estático
- Los literales de la serie de SQL y los identificadores delimitados se pueden continuar 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, para seleccionar datos de la columna NAME de la tabla staff donde la columna NAME sea igual a 'Sanders', podría hacer algo parecido a lo siguiente:

  ```
  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 que se pasa al gestor de base de datos como sentencia de SQL.
- 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 y Linux utilizan un carácter de salto de línea.

Sentencias de SQL incorporado en aplicaciones FORTRAN

Las sentencias de SQL incorporado en aplicaciones FORTRAN constan de estos tres elementos:

Sintaxis correcta de elementos en FORTRAN

Inicializador de la sentencia
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 reglas siguientes a las sentencias de SQL incorporado en aplicaciones FORTRAN:

- 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 inicializador de sentencia EXEC SQL en distintas líneas.
- Están permitidos los comentarios de SQL en cualquier línea que forme parte de una sentencia de SQL incorporado. Estos comentarios no están permitidos en las sentencias que se ejecutan de forma dinámica. El formato de un comentario de SQL consiste en un guión doble (--), seguido de una serie compuesta por cero o más caracteres y terminada por un fin de línea.
- Los comentarios FORTRAN están permitidos *casi* en cualquier lugar de una sentencia de SQL incorporada. Las excepciones son:
  – No se permiten comentarios entre EXEC y SQL.
  – No se permiten comentarios en las sentencias que se ejecutan dinámicamente.
  – La ampliación de utilizar ! para codificar un comentario FORTRAN al final de una línea no recibe soporte dentro de una sentencia de SQL incorporado.

- Utilice una notación exponencial cuando especifique una constante real en las sentencias de SQL. El gestor de bases de datos interpreta una serie de dígitos con una coma decimal en una sentencia de SQL como una constante decimal, no como una constante real.
- Los números de sentencia no son válidos en las sentencias de SQL que preceden a la primera sentencia FORTRAN ejecutable. Si una sentencia de SQL tiene asociado un número de sentencia, el precompilador genera una sentencia CONTINUE etiquetada que precede directamente a la sentencia de SQL.
- Cuando haga referencia a variables del lenguaje principal dentro de una sentencia de SQL, utilícelas exactamente tal como se han declarado.
- La sustitución de caracteres de espacio en blanco, como caracteres de fin de línea y de tabulación, se produce del siguiente modo:
  - Cuando aparecen fuera de las comillas (pero dentro de sentencias de SQL), los caracteres de fin de línea y tabuladores se sustituyen por un solo espacio.
  - Si aparecen dentro de las comillas, los caracteres de fin de línea desaparecen, siempre que se continúe correctamente la serie para un programa FORTRAN. Los tabuladores no se modifican.

  Observe que los caracteres reales utilizados como fin de línea y tabulador varían en función de la plataforma. Por ejemplo, las plataformas basadas en Windows utilizan el retorno de carro/salto de línea como fin de línea, mientras que las plataformas basadas en UNIX y Linux sólo utilizan simplemente un salto de línea.

Sentencias de SQL incorporado en aplicaciones COBOL

Las sentencias de SQL incorporado en aplicaciones COBOL constan de estos tres elementos:

Sintaxis correcta de elementos en COBOL

Inicializador de la sentencia
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 reglas siguientes a las sentencias de SQL incorporado en aplicaciones COBOL:

- Las sentencias de SQL ejecutables se deben colocar en la sección `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 el inicializador de sentencia EXEC SQL y termínela con el terminador de sentencia 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.
- Se permiten comentarios de COBOL en casi todas partes. 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 inicializador de sentencia 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 importar los archivos de inclusión.
- 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.
- La sustitución de caracteres de espacio en blanco, como caracteres de fin de línea y de tabulación, se produce del siguiente modo:
  - Cuando aparecen fuera de las comillas (pero dentro de sentencias de SQL), los caracteres de fin de línea y tabuladores se sustituyen por un solo espacio.
  - Si aparecen dentro de las comillas, los caracteres de fin de línea desaparecen, siempre que se continúe correctamente la serie para un programa COBOL. Los tabuladores no se modifican.

  Observe que los caracteres reales utilizados como fin de línea y tabulador varían en función de la plataforma. Por ejemplo, las plataformas basadas en Windows utilizan el retorno de carro/salto de línea como fin de línea, mientras que los sistemas basados en UNIX y Linux utilizan simplemente un salto de línea.

Sentencias de 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 interpretado, resultará más fácil desarrollar y depurar los prototipos de aplicación en REXX que en lenguajes principales compilados. Aunque las aplicaciones de bases de datos codificadas en REXX no proporcionan el

rendimiento de las aplicaciones de bases de datos que utilizan lenguajes compilados, proporcional la posibilidad de crear aplicaciones de bases de datos 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 'texto SQL',
             'texto adicional',
                  .
                  .
                  .
             'texto final'
```

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: en aplicaciones REXX
- 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.

Nota: REXX no da soporte al acceso a bases de datos de varias hebras.

Capítulo 2. Software de desarrollo soportado para aplicaciones de SQL incorporado

Los sistemas de bases de datos DB2 dan soporte a compiladores, intérpretes y software de desarrollo relacionado para aplicaciones de SQL incorporado en los siguientes sistemas operativos:

- AIX
- HP-UX
- Linux
- Solaris
- Windows

Las aplicaciones de SQL incorporado de 32 y de 64 bits se pueden crear a partir de código fuente de SQL incorporado.

Los siguientes lenguajes principales requieren compiladores específicos para desarrollar aplicaciones de SQL incorporado:

- C
- C++
- COBOL
- Fortran
- REXX

Capítulo 3. Configuración del entorno de desarrollo de SQL incorporado

Para poder empezar a crear aplicaciones de SQL incorporado, tiene que instalar el compilador soportado para el lenguaje principal que va a utilizar para desarrollar las aplicaciones y configurar el entorno de SQL incorporado.

- Servidor de bases de datos DB2 instalado en una plataforma soportada
- Cliente DB2 instalado
- Software de desarrollo de aplicaciones de SQL incorporado soportado instalado - vea "Software de desarrollo de aplicaciones de SQL incorporado soportado instalado" en *Iniciación al desarrollo de aplicaciones de bases de datos*

Asigne al usuario la autorización para emitir el mandato PREP y el mandato BIND.

Para verificar que el entorno de desarrollo de aplicaciones de SQL incorporado está correctamente configurado, intente crear y ejecutar la plantilla de aplicación de SQL incorporado que encontrará en el tema: Plantilla de aplicación de SQL incorporado en C.

Capítulo 4. Diseño de aplicaciones de SQL incorporado

Cuando se diseñan aplicaciones de SQL incorporado, se tienen que utilizar sentencias de SQL ejecutadas estática o dinámicamente. Las sentencias de SQL estático son de dos tipos: sentencias que no contienen variables del lenguaje principal (utilizadas principalmente para inicialización y ejemplos de SQL sencillos) y sentencias que utilizan variables del lenguaje principal. Las sentencias de SQL dinámico también son de dos tipos: pueden no contener ningún marcador de parámetro (típico de interfaces como CLP) o pueden contener marcadores de parámetro, lo que ofrece mayor flexibilidad dentro de las aplicaciones.

La opción sobre si utilizar sentencias ejecutadas estática o dinámicamente depende de varios factores que incluyen portabilidad, rendimiento y restricciones de las aplicaciones de SQL incorporado.

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).

Los grupos proporcionan un medio conveniente para llevar a cabo la autorización para un conjunto de usuarios sin tener que otorgar ni revocar privilegios individualmente para cada usuario. La pertenencia a un grupo se tiene en cuenta en la ejecución de sentencias de SQL dinámico, pero no en sentencias de SQL estático. Sin embargo, los privilegios `PUBLIC` se tienen en cuenta en la ejecución de sentencias de SQL estático. Por ejemplo, suponga que tiene un procedimiento almacenado de SQL incorporado con consultas de SQL vinculadas estáticamente sobre una tabla llamada `STAFF`. Si intenta crear este procedimiento con la sentencia `CREATE PROCEDURE` y la cuenta pertenece a un grupo que tiene el privilegio de selección para la tabla `STAFF`, la sentencia `CREATE` fallará con un error SQL0551N. Para que la sentencia `CREATE` funcione, la cuenta necesita directamente el privilegio de selección en la tabla `STAFF`.

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.

Los grupos proporcionan un medio conveniente para llevar a cabo la autorización para un conjunto de usuarios sin tener que otorgar ni revocar privilegios individualmente para cada usuario. En general, la pertenencia a un grupo se tiene en cuenta en sentencias de SQL dinámico, pero no se tiene en cuenta en sentencias de SQL estático. La excepción a este caso general se da cuando se otorgan privilegios a PUBLIC: estos privilegios se tienen en cuenta cuando se procesan sentencias de SQL.

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.

Ejecución de sentencias de SQL estático y dinámico en aplicaciones de SQL incorporado

La ejecución tanto de sentencias de SQL estático y como de SQL dinámico recibe soporte en aplicaciones de SQL incorporado. Para decidir si hay que ejecutar las sentencias de SQL de forma estática o dinámica hay que comprender los paquetes, cómo se ejecutan las sentencias de SQL en el momento de la ejecución, las variables del lenguaje principal, los marcadores de parámetro y cómo estos elementos se relacionan con el rendimiento de la aplicación.

SQL estático en programas de SQL incorporado

A continuación se muestra un ejemplo de una sentencia ejecutada estáticamente en C:

```
/* seleccionar valores de table a variables lenguaje principal */
/*  con STATIC SQL e imprimirlos */
EXEC SQL SELECT id, name, dept, salary INTO :id, :name, :dept, :salary
           FROM staff WHERE id = 310;
printf("
```

SQL dinámico en programas de SQL incorporado

A continuación se muestra un ejemplo de una sentencia ejecutada dinámicamente en C:

```
/* Actualizar columna de tabla mediante DYNAMIC SQL*/
strcpy(hostVarStmtDyn, "UPDATE staff SET salary = salary + 1000  WHERE dept = ?");
EXEC SQL PREPARE StmtDyn FROM :hostVarStmtDyn;
EXEC SQL EXECUTE StmtDyn USING :dept;
```

Sentencias dinámicas de SQL incorporado

Las sentencias 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.

Se necesitan sentencias de soporte de SQL dinámico para transformar la variable del lenguaje principal que contiene texto de SQL en un formato ejecutable. Además, las sentencias de soporte de SQL dinámico funcionan en la variable del lenguaje principal haciendo referencia al nombre de la sentencia. Estas sentencias de soporte son:

EXECUTE IMMEDIATE
Prepara y ejecuta una sentencia que no utiliza ninguna variable del lenguaje principal. Utilice esta sentencia como alternativa a las sentencias PREPARE y EXECUTE.

Por ejemplo, considere la siguiente sentencia en C:

```
strcpy (qstring,"INSERT INTO WORK_TABLE SELECT *
   FROM EMP_ACT WHERE ACTNO >= 100");
EXEC SQL  EXECUTE IMMEDIATE :qstring;
```

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.

Por ejemplo, considere la siguiente sentencia en C:

```
strcpy(hostVarStmt, "DELETE FROM org WHERE deptnumb = 15");
EXEC SQL PREPARE Stmt FROM :hostVarStmt;
EXEC SQL DESCRIBE Stmt INTO :sqlda;
EXEC SQL EXECUTE Stmt;
```

Nota: El contenido de las sentencias de SQL dinámico sigue la misma sintaxis que las sentencias de SQL estático, con las siguientes excepciones:

- 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 (;).

Determinación del momento en que deben ejecutarse sentencias de SQL estática o dinámicamente en aplicaciones de SQL incorporado

Hay algunas consideraciones que deben tenerse en cuenta antes de determinar si una sentencia de SQL se ejecuta estática o dinámicamente en una aplicación de SQL incorporado. En las tablas siguientes se listan las consideraciones más importantes a tener en cuenta, así como las recomendaciones relativas a cuándo se debe utilizar SQL estático o dinámico, o bien en qué casos ambas opciones son igualmente adecuadas.

Nota: Esta tabla contiene recomendaciones generales. El requisito de la aplicación, 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 comparar las diferencias.

Tabla 1. Comparación entre SQL estático y dinámico

Consideraciones	**Probablemente la mejor opción**
Tiempo deseado en el que debe ejecutarse la consulta:	
• Menos de 2 segundos	• Estático
• Entre 2 y 10 segundos	• Cualquiera
• Más de 10 segundos	• Dinámico
Uniformidad de datos que se consultan o con los que se trabaja por parte de la sentencia de SQL	
• Distribución de datos uniforme	• Estático
• Ligera falta de uniformidad	• Cualquiera
• Distribución nada uniforme	• Dinámico
Cantidad de predicados de rango en la consulta	
• Pocos	• Estático
• Algunos	• Cualquiera
• Muchos	• Dinámico
Posibilidad de la ejecución de sentencias de SQL repetidas	
• 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
Tipos de sentencias de SQL (DML/DDL/DCL)	
• 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 con la que se emite el mandato `RUNSTATS`	
• Muy poco frecuente	• Estático
• Regular	• Cualquiera
• Frecuente	• Dinámico

Las sentencias de SQL siempre se compilan antes de ejecutarlas. La diferencia estriba en que las sentencias de SQL dinámico se compilan en tiempo de ejecución; por eso, la aplicación puede ser un poco más lenta debido a la actividad general de compilar cada sentencia dinámica en tiempo de ejecución de la aplicación, en comparación con una única fase de compilación inicial, como sucede en el caso de SQL estático.

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, SQL dinámico es 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.

En algunas ocasiones no importará mucho si se utiliza SQL estático o dinámico. Por ejemplo, podría ser el caso de una aplicación que contenga básicamente referencias a sentencias de SQL que deben ejecutarse dinámicamente, aunque podría haber una sentencia que sería más adecuado ejecutarla como SQL estático. En ese caso, para ser coherente con el código, podría tener más sentido simplemente ejecutar también esa sentencia dinámicamente. Observe que las consideraciones de la tabla anterior se listan en líneas generales por orden de importancia.

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

Nota: Tanto SQL estático como SQL dinámico vienen en dos tipos, sentencias que utilizan variables del lenguaje principal y otras que no. Estos tipos son:

1. Sentencias de SQL estático que no contiene variables del lenguaje principal

   Esta es una situación poco probable que sólo verá para:

   - Código de inicialización
   - Sentencias de SQL muy simples

   Las sentencias de SQL simples sin variables del lenguaje principal tienen un buen rendimiento 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

   Las sentencias de SQL estático que utilizan variables del lenguaje principal se consideran como el estilo tradicional de las aplicaciones de DB2. Las sentencias de SQL estático evitan 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

Esto es normal en interfaces como CLP, que suele utilizarse para ejecutar consultas para un propósito concreto. Desde la interfaz CLP, las sentencias de SQL sólo se pueden ejecutar dinámicamente.

4. SQL dinámico que contiene marcadores de parámetros

   La ventaja clave de las sentencias de SQL dinámico es que la presencia de marcadores de parámetros permite amortizar el coste de la preparación de la sentencia 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.

Rendimiento de aplicaciones de SQL incorporado

El rendimiento es un factor importante a tener en cuenta cuando se desarrollan aplicaciones de bases de datos. Las aplicaciones de SQL estático se pueden ejecutar bien, porque dan soporte a la ejecución de sentencias de SQL estático y a una combinación de ejecución de sentencias de SQL estático y dinámico. Debido a la forma en que se compilan las sentencias de SQL estático, hay pasos que un desarrollador o administrador de bases de datos debe seguir para asegurarse de que las aplicaciones de SQL incorporado continúan ejecutándose bien con el tiempo.

Los siguientes factores pueden afectar al rendimiento de las aplicaciones de SQL incorporado:

- Cambios en esquemas de bases de datos con el tiempo
- Cambios en las cardinalidades de las tablas (el número de filas de las tablas) con el tiempo
- Cambios en los valores de las variables del lenguaje principal vinculadas a sentencias de SQL

El rendimiento de las aplicaciones de SQL incorporado se ve afectado por estos factores porque el paquete se crea una vez cuando una base de datos puede tener un determinado conjunto de características. Estas características intervienen en la creación de los planes de acceso de tiempo de ejecución de paquetes, que definen la forma en que el gestor de bases de datos ejecutará de la forma más eficiente posible las sentencias de SQL. Con el tiempo, un esquema de base de datos y los datos pueden cambiar, lo que hace que los planes de acceso de tiempo de ejecución dejen de ser los óptimos. Esto puede dar lugar a una degradación del rendimiento de la aplicación.

Por este motivo, es importante renovar periódicamente la información que se utiliza para asegurar que los planes de acceso de tiempo de ejecución de los paquetes se mantienen correctamente.

Se utiliza el mandato `RUNSTATS` para recopilar estadísticas actuales sobre tablas e índices, especialmente si se ha producido una actividad de actualización significativa o se han creado nuevos índices desde la última vez que se ejecutó el mandato `RUNSTATS`. Esto proporciona al optimizador la información más precisa con la que puede determinar el mejor plan de acceso.

El rendimiento de las aplicaciones de SQL incorporado se puede mejorar de varias formas:

- Ejecutando el mandato RUNSTATS para actualizar estadísticas de base de datos.
- Volviendo a vincular paquetes de aplicación con la base de datos para volver a generar los planes de acceso de tiempo de ejecución (basados en las estadísticas actualizadas) que utilizará la base de datos para recuperar físicamente los datos en disco.
- Utilizando la opción de enlace REOPT en los programas estáticos y dinámicos.

Soporte de 32 bits y 64 bits para aplicaciones de SQL incorporado

Las aplicaciones de SQL incorporado se pueden crear en plataformas de 32 y de 64 bits. No obstante, existen consideraciones separadas sobre la creación y la ejecución. Los scripts de construcción contienen una comprobación para determinar el ancho de bits. Si el ancho de bits detectado es de 64 bits, se establece un conjunto de conmutadores adicional para acomodar los cambios necesarios.

Los sistemas de base de datos DB2 están soportados en versiones de 32 y 64 bits de los sistemas operativos listados más abajo. En la mayoría de estos sistemas operativos existen diferencias respecto a la forma de crear aplicaciones de SQL incorporado en los entornos de 32 y 64 bits.

- AIX
- HP-UX
- Linux
- Solaris
- Windows

Las únicas instancias de 32 bits que estarán soportadas en DB2 Versión 9 son:

- Linux en x86
- Windows en x86
- Windows en X64 (al utilizar la imagen de instalación de DB2 para Windows en x86)

Las únicas instancias de 64 bits que estarán soportadas en DB2 Versión 9 son:

- AIX
- Sun
- HP PA-RISC
- HP IPF
- Linux en x86_64
- Linux en POWER
- Linux en zSeries
- Windows en X64 (al utilizar la imagen de instalación de Windows para X64)
- Windows en IPF
- Linux en IPF

Los sistemas de base de datos DB2 dan soporte a la ejecución de aplicaciones y rutinas de 32 bits en todos los entornos de sistema operativo de 64 bits soportados excepto Linux IA64 y Linux zSeries.

Para cada uno de los lenguajes principales, las variables del lenguaje principal utilizadas pueden ser mejor en plataformas de 32 bits o 64 bits o ambas. Compruebe los diversos tipos de datos para cada uno de los lenguajes de programación.

Restricciones en aplicaciones SQL incorporadas

Cada lenguaje principal soportado tiene su propio conjunto de limitaciones y especificaciones. C/C++ utiliza una secuencia de tres caracteres denominada trígrafo para superar las limitaciones de visualización de determinados caracteres especiales. COBOL tiene un conjunto de normas que facilitan la utilización de aplicaciones COBOL orientadas a objetos. FORTRAN tiene áreas de interés que pueden afectar a los procesos de precompilación, mientras que REXX está limitado a determinadas áreas, como por ejemplo el soporte de lenguajes.

Restricciones sobre juegos de caracteres cuando se utiliza C y C++ para programar aplicaciones de SQL incorporado

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**
??=	Marca de símbolo numérico '#'
??/	Barra invertida '\'
??'	Signo '^'
??!	Barra vertical '|'
??–	Tilde '~'

Restricciones en el uso de COBOL para programar aplicaciones de SQL incorporado

A continuación se describen las restricciones correspondientes a llamadas a API en aplicaciones COBOL:

- Todas las variables enteras utilizadas como parámetros de valor en las llamadas a las API se deben declarar con una cláusula `USAGE COMP-5`.

En un programa COBOL orientado a objetos:

- 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.
- 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.

Restricciones en el uso de FORTRAN para programar aplicaciones de SQL incorporado

El soporte de SQL incorporado de FORTRAN se ha estabilizado en DB2 Universal Database Versión 5 y no se ha planificado ninguna mejora para el futuro. Por ejemplo, el precompilador FORTRAN no puede manejar identificadores de objetos SQL, como por ejemplo, los nombres de tabla, que tengan una longitud superior a 18 bytes. Para utilizar las características incorporadas en los sistemas de bases de datos DB2 después de UDB 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.

El desarrollo de aplicaciones de bases de datos FORTRAN no está soportado en instancias de DB2 en entornos Windows o Linux.

FORTRAN no da soporte al acceso a bases de datos de varias hebras.

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.

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.

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`.

Restricciones en el uso de REXX para programar aplicaciones de SQL incorporado

A continuación se describen las restricciones correspondientes a SQL incorporado en aplicaciones REXX:

- El soporte de SQL incorporado de REXX se ha estabilizado en DB2 Universal Database Versión 5 y no se ha planificado ninguna mejora para el futuro. Por ejemplo, REXX no puede manejar identificadores de objetos SQL, como por ejemplo, los nombres de tabla, que tengan una longitud superior a 18 bytes. Para utilizar las características incorporadas en los sistemas de bases de datos 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.
- En REXX/SQL no se soporta el SQL compuesto.

- REXX no da soporte a SQL estático.
- Las aplicaciones REXX no se soportan en los entornos EUC en japonés o chino tradicional.

Recomendaciones para desarrollar aplicaciones de SQL incorporado con XML y XQuery

Las siguientes recomendaciones son válidas para usar XML y XQuery en aplicaciones de SQL incorporado.

- Las aplicaciones deben acceder a todos los datos XML en formato de serie serializada.
  - Debe representar todos los datos, incluidos los datos numéricos y de fecha y hora, en su formato de serie serializado.
- Los datos XML externalizados están limitados a 2 GB.
- Todos los cursores que contengan datos XML son de no bloqueo (cada operación de captación genera una petición del servidor de bases de datos).
- Cuando las variables del lenguaje principal de tipo carácter contienen datos XML serializados, se da por supuesto que se utiliza la página de códigos de la aplicación como la codificación de los datos y debe coincidir con cualquier codificación interna existente en los datos.
- Debe especificar un tipo de datos LOB como el tipo base para una variable del lenguaje principal XML.
- Se aplica lo siguiente a SQL estático:
  - No se pueden utilizar variables del lenguaje principal de tipo carácter y binarias para recuperar valores XML de una operación `SELECT INTO`.
  - Si se espera un tipo de datos XML como entrada, el uso de las variables de lenguaje principal CHAR, VARCHAR, CLOB y BLOB estará sujeto a una operaciónn `XMLPARSE` con las características por omisión del manejo de espacios en blanco (`'STRIP WHITESPACE'`). Cualquier otro tipo de variable del lenguaje principal que no sea XML se rechazará.
  - No hay soporte para las expresiones de XQuery estático; si se intenta precompilar una expresión XQuery, la operación fallará con un error. Sólo puede ejecutar expresiones XQuery a través de la función XMLQUERY.
- Una expresión XQuery se puede ejecutar dinámicamente añadiendo a la expresión la serie ″XQUERY″ como prefijo.

Transacciones simultáneas y acceso a base de datos de varias hebras en aplicaciones de SQL incorporado

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 DB2 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 de SQL incorporado 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. El desarrollo de aplicaciones de SQL incorporado de varias hebras con código con protección de hebras sólo recibe soporte en C y C++. Puede escribir su propio compilador, que junto con las funciones que proporciona el lenguaje permita el acceso simultáneo a bases de datos de varias hebras.

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.

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.

En la implementación por omisión de las aplicaciones de hebras sobre una base de datos DB2, la serialización del acceso a la base de datos se impone mediante las API de la base de datos. Si una hebra realiza una llamada a la base de datos, las llamadas realizadas por otras hebras se bloquearán hasta que finalice la primera llamada, aunque las siguientes llamadas accedan a objetos de la base de datos que no estén relacionados con la primera llamada. Además, todas las hebras de un proceso comparten un ámbito de confirmación. El verdadero acceso simultáneo a una base de datos sólo se puede conseguir mediante procesos separados o utilizando las API que se describen en este tema.

Los sistemas de bases de datos DB2 proporcionan API que se pueden utilizar para asignar y manipular entornos separados (contextos) para el uso de las API de la base de datos y SQL incorporado. Cada contexto es una entidad separada y cualquier conexión o enlace realizado mediante un contexto es independiente de los demás contextos (por lo tanto, de las demás conexiones o enlaces de un proceso). Para poder trabajar sobre un contexto, primero se debe asociar a una hebra. Una hebra siempre debe tener un contexto cuando se realizan llamadas a las API de la base de datos o cuando se utiliza SQL incorporado.

Por omisión, todas las aplicaciones del sistema de bases de datos DB2 son de varias hebras y pueden utilizar varios contextos. Puede utilizar las siguientes API de DB2 para utilizar varios contextos. Específicamente, su aplicación puede crear un contexto para una hebra, conectarse o desconectarse de un contexto independiente para cada hebra y pasar contextos entre las hebras. Si su aplicación no llama a *ninguna* de estas API, DB2 administrará automáticamente los contextos múltiples para sus aplicaciones:

- `sqleAttachToCtx` - Enlazar a contexto
- `sqleBeginCtx` - Crear y enlazar a un contexto de aplicación
- `sqleDetachFromCtx` - Desenlazar de contexto
- `sqleEndCtx` - Desenlazar y destruir contexto de aplicación
- `sqleGetCurrentCtx` - Obtener contexto actual
- `sqleInterruptCtx` - Interrumpir contexto

Esas API no tienen efecto (es decir, son no-ops) sobre las plataformas que no dan soporte a hebras de aplicaciones.

Los contextos se tienen que asociar a una determinada hebra durante el tiempo que dure una conexión o enlace. Una hebra puede enlazarse a un contexto, conectarse a una base de datos, desenlazarse del contexto y luego una segunda hebra se puede enlazar al contexto y continuar trabajando utilizando la conexión de base de datos existente. Los contextos se pueden pasar entre hebras de un proceso, pero no entre procesos.

Aunque se utilicen las nueva API, las siguientes API continúan estando serializadas:

- `sqlabndx` - Vincular
- `sqlaprep` - Precompilar programa
- `sqluexpr` - Exportar
- `db2Import` y `sqluimpr` - Importar

Nota:

1. La CLI de DB2 utiliza automáticamente múltiples contextos para conseguir acceso con protección de hebras y simultáneo a la base de datos en las plataformas que dan soporte a varias hebras. Aunque no resulta recomendable para DB2, los usuarios pueden inhabilitar de forma explícita esta función si lo desean.
2. Por omisión, AIX no permite que las aplicaciones de 32 bits se enlacen a más de 11 segmentos de memoria compartida por proceso, de los cuales un máximo de 10 se pueden utilizar para conexiones de DB2.

   Cuando se alcanza este límite, DB2 devuelve SQLCODE -1224 en SQL CONNECT. DB2 Connect también tiene un límite de 10 conexiones si los usuarios locales ejecutan la confirmación ene dos con un supervisor de TP (TCP/IP).

   La variable de entorno AIX `EXTSHM` se puede utilizar para aumentar el número máximo de segmentos de memoria compartida a los que se puede enlazar un proceso.

   Para utilizar `EXTSHM` con DB2, haga lo siguiente:

   En sesiones cliente:

   ```
   export EXTSHM=ON
   ```

   Cuando se inicie el servidor de DB2:

   ```
   export EXTSHM=ON
   db2set DB2ENVLIST=EXTSHM
   db2start
   ```

   En DPF, añada también las siguientes líneas a los archivos userprofile o usercshrc:

   ```
   EXTSHM=ON
   export EXTSHM
   ```

Una alternativa consiste en mover la base de datos local o DB2 Connect a otra máquina y acceder a la misma de forma remota, o acceder a la base de datos de DB2 Connect con retorno de bucle TCP/IP, catalogándolo como un nodo remoto que tiene la dirección TCP/IP de la máquina local.

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 la página de códigos y el código de país o región para aplicaciones UNIX de varias hebras

Esta sección es específica de las aplicaciones de SQL incorporado C y C++.

En AIX, Solaris y HP-UX, las funciones que se utilizan para las consultas en tiempo de ejecución de la página de códigos y del código de país o región que se van a utilizar para una conexión de base de datos tienen ahora protección de hebras. Pero estas funciones pueden crear cierta contención de bloqueo (lo que da lugar a una degradación del rendimiento) en una aplicación de varias hebras que utilice un gran número de conexiones simultáneas de base de datos.

Puede utilizar la variable de entorno *DB2_FORCE_NLS_CACHE* para eliminar la posibilidad de que se produzca contención de bloqueo en aplicaciones de varias hebras. Cuando *DB2_FORCE_NLS_CACHE* tiene el valor `TRUE`, la información sobre página de códigos y código de país o región se guarda la primera vez que una hebra accede a la misma. 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.

Resolución de problemas de aplicaciones de SQL incorporado de varias hebras

Las secciones siguientes describen problemas que se pueden producir con aplicaciones de SQL incorporado de varias hebras y cómo evitarlos.

Una aplicación que utiliza varias hebras es 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 puede detectar el punto muerto, uno de los contextos recibirá SQLCODE -911 y su unidad de trabajo se retrotraerá.

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.

Prevención de 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

context 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.

Capítulo 5. Programación de aplicaciones de SQL incorporado

La programación de aplicaciones de SQL incorporado implica todos los pasos necesarios para ensamblar una aplicación en el lenguaje elegido de programación de SQL incorporado. Una vez determine que SQL incorporado es la API adecuada para sus necesidades de programación, y tras diseñar la aplicación de SQL incorporado, estará preparado para programar una aplicación de SQL incorporado.

Requisitos previos:

- Elección de si desea utilizar sentencias de SQL estático o dinámico
- Diseño de una aplicación de SQL incorporado

La programación de aplicaciones de SQL incorporado consta de las siguientes subtareas:

- Inclusión de los archivos de cabecera necesarios
- Elección de un lenguaje de programación de SQL incorporado soportado
- Declaración de variables del lenguaje principal para representar los valores que se van a incluir en sentencias de SQL
- Conectar a una fuente de datos utilizando JDBC
- Ejecución de sentencias de SQL en aplicaciones de SQL incorporado
- Manejo de errores y avisos de SQL relacionados con la ejecución de sentencias de SQL
- Desconexión de la fuente de datos

Cuando haya completado la aplicación de SQL incorporado, estará listo para compilar y ejecutar la aplicación: Creación de aplicaciones de SQL incorporado.

Archivos fuente de SQL incorporado

Cuando desarrolla código fuente que incluye SQL incorporado, tiene que seguir ciertos convenios de denominación de archivos específicos para cada uno de los lenguajes principales soportados.

Archivos de entrada y de salida para C y C++

Por omisión, la aplicación fuente puede tener las siguientes extensiones:

.sqc Para archivos C en todos los sistemas operativos soportados

.sqC Para archivos C++ en los sistemas operativos UNIX y Linux

.sqx Para archivos C++ en sistemas operativos Windows.

Por omisión, los archivos de salida del precompilador correspondiente tienen las siguientes extensiones:

.c Para archivos C en todos los sistemas operativos soportados

.C Para archivos C++ en los sistemas operativos UNIX y Linux

.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 de entrada y salida para COBOL

Por omisión, la aplicación fuente tiene la extensión:

.sqb Para archivos COBOL en todos los sistemas operativos

Sin embargo, si utiliza la opción de precompilación TARGET, (TARGET ANSI_COBOL, TARGET IBMCOB o TARGET MFCOB), el archivo de entrada puede tener la extensión que prefiera.

Por omisión, los archivos de salida del precompilador correspondiente tienen las siguientes extensiones:

.cbl Para archivos COBOL en todos los sistemas operativos

Sin embargo, puede utilizar la opción de precompilación OUTPUT para especificar un nuevo nombre y vía de acceso para el archivo fuente modificado de salida.

Archivos de entrada y salida para FORTRAN

Por omisión, la aplicación fuente tiene la extensión:

.sqf Para archivos FORTRAN en todos los sistemas operativos

Sin embargo, si utiliza la opción de precompilación TARGET con la opción FORTRAN, el archivo de entrada puede tener la extensión que prefiera.

Por omisión, los archivos de salida del precompilador correspondiente tienen las siguientes extensiones:

.f Para archivos FORTRAN en los sistemas operativos UNIX y Linux

.for Para archivos FORTRAN en sistemas operativos Windows

Sin embargo, puede utilizar la opción de precompilación OUTPUT para especificar un nuevo nombre y vía de acceso para el archivo fuente modificado de salida.

Plantilla de aplicación de SQL incorporado en C

Esta es una aplicación de SQL incorporado simple que se le proporciona para probar el entorno de desarrollo de SQL incorporado y como ayuda para aprender la estructura básica de las aplicaciones de SQL incorporado.

Las aplicaciones de SQL incorporado requieren la siguiente estructura:

- inclusión de los archivos de cabecera necesarios
- declaraciones de variables del lenguaje principal para los valores que se van a incluir en sentencias de SQL
- una conexión de base de datos
- la ejecución de sentencias de SQL
- el manejo de errores y avisos de SQL relacionados con la ejecución de sentencias de SQL
- desconexión de la conexión de base de datos

El siguiente código fuente demuestra la estructura básica necesaria para las aplicaciones de SQL incorporado escritas en C.

Programa de ejemplo: template.sqc

```
 #include <stdio.h>                                                    1
 #include <stdlib.h>
 #include <string.h>
 #include <sqlenv.h>
 #include <sqlutil.h>

EXEC SQL BEGIN DECLARE SECTION;                                        2
  short id;
  char name[10];
  short dept;
  double    salary;
  char hostVarStmtDyn[50];
EXEC SQL END DECLARE SECTION;

int main()
 {
   int rc = 0;                                                         3
   EXEC SQL INCLUDE SQLCA;                                             4

   /* conectar con base de datos */
   printf("\n Connecting to database...");
   EXEC SQL CONNECT TO "sample";                                       5
   if (SQLCODE  <0)                                                    6
   {
      printf("\nConnect Error:  SQLCODE =
      goto connect_reset;
   }
   else
   {
      printf("\n Connected to database.\n");
   }

   /* ejecutar una sentencia SQL (consulta) mediante SQL estático; copiar
     la fila de valores de resultado a variables del lenguaje principal*/
   EXEC SQL SELECT id, name, dept, salary                              7
           INTO :id, :name, :dept, :salary
           FROM staff WHERE id = 310;
   if (SQLCODE  <0)                                                    6
   {
      printf("Select Error:  SQLCODE =
   }
   else
   {
      /* imprimir los valores de variable del lenguaje principal a
         salida estándar */
      printf("\n Executing a static SQL query statement, searching for
         \n the id value equal to 310\n");
      printf("\n ID   Name        DEPT       Salary\n");
      printf("
   }

   strcpy(hostVarStmtDyn, "UPDATE staff
                          SET salary = salary + 1000
                          WHERE dept = ?");
   /* ejecutar una sentencia de SQL (operación) utilizando una variable del
      lenguaje principal y DYNAMIC SQL*/
   EXEC SQL PREPARE StmtDyn FROM :hostVarStmtDyn;
   if (SQLCODE  <0)                                                    6
   {
      printf("Prepare Error:  SQLCODE =
   }
   else
```

```
{
   EXEC SQL EXECUTE StmtDyn USING :dept;                              8
}
if (SQLCODE  <0)                                                      6
{
   printf("Execute Error:  SQLCODE =
}

/* Leer la fila actualizada mediante STATIC SQL y CURSOR */
EXEC SQL DECLARE posCur1 CURSOR FOR
   SELECT id, name, dept, salary
   FROM staff WHERE id = 310;
if (SQLCODE  <0)                                                      6
{
   printf("Declare Error:  SQLCODE =
}
EXEC SQL OPEN posCur1;
EXEC SQL FETCH posCur1 INTO :id, :name, :dept, :salary ;              9
if (SQLCODE  <0)                                                      6
{
   printf("Fetch Error:  SQLCODE =
}
else
{
   printf(" Executing an dynamic SQL statement, updating the
            \n salary value for the id equal to 310\n");
   printf("\n ID   Name        DEPT       Salary\n");
   printf("
}

EXEC SQL CLOSE posCur1;

/* Confirmar la transacción */
printf("\n  Commit the transaction.\n");
EXEC SQL COMMIT;                                                      10
if (SQLCODE  <0)                                                      6
{
   printf("Error:  SQLCODE =
}

/* Desconectar de la base de datos */
connect_reset :
   EXEC SQL CONNECT RESET;                                            11
   if (SQLCODE  <0)                                                   6
   {
      printf("Connection Error:  SQLCODE =
   }
return 0;
} /* end main */
```

Notas para el **Programa de ejemplo: template.sqc**:

Nota	Descripción
1	Incluir archivos: esta directriz incluye un archivo en la aplicación fuente.
2	Sección de declaración: la declaración de variables del lenguaje principal que se utilizarán para contener valores a los que se hace referencia en las sentencias de SQL de la aplicación C.
3	Declaración de variable local: este bloque declara las variables locales a utilizar en la aplicación. Estas no son variables del lenguaje principal.
4	Incluyendo la estructura SQLCA: la estructura SQLCA se actualiza tras la ejecución de cada sentencia de SQL. Esta aplicación de plantilla utiliza determinados campos de SQLCA para el manejo de errores.

Nota	Descripción
5	Conexión a una base de datos: el paso inicial al trabajar con la base de datos es establecer una conexión a la base de datos. Aquí, la conexión se efectúa ejecutando la sentencia de SQL `CONNECT`.
6	Manejo de errores: comprueba si se ha producido un error.
7	Ejecutar una consulta: la ejecución de esta sentencia de SQL asigna datos devueltos de una tabla a variables del lenguaje principal. El código C debajo de la ejecución de la sentencia de SQL imprime los valores de las variables del lenguaje principal en salida estándar.
8	Ejecutar una operación: la ejecución de esta sentencia de SQL actualiza un conjunto de filas de una tabla identificada por su número de departamento. La preparación (realizada tres líneas antes) es un paso en el que los valores de variables del lenguaje principal, como a la que se hace referencia en esta sentencia, están enlazadas a la sentencia de SQL que debe ejecutarse.
9	Ejecutar una operación: en esta línea y en la línea anterior, esta aplicación utiliza cursores en SQL estático para seleccionar información de una tabla e imprimir los datos. Una vez se ha declarado y abierto el cursor, se cargan los datos y, finalmente, se cierra el cursor.
10	Confirmar la transacción: la sentencia COMMIT finaliza los cambios de la base de datos que se realizaron dentro de una unidad de trabajo.
11	Finalmente, la conexión de base de datos debe desactivarse.

Archivos de inclusión y definiciones necesarios para aplicaciones de SQL incorporado

Los archivos de inclusión son necesarios para proporcionar funciones y tipos utilizados dentro de la biblioteca. Deben incluirse para que el programa pueda utilizar las funciones de biblioteca. Por omisión, estos archivos se instalarán en la carpeta $HOME/sqllib/include. Cada lenguaje principal tiene sus propios métodos para incluir archivos, así como para utilizar distintas extensiones de archivo. Dependiendo del lenguaje especificado, deben tomarse ciertas precauciones, como especificar vías de acceso de archivo.

Archivos de inclusión para aplicaciones de SQL incorporado C y C++

Los archivos de inclusión específicos del lenguaje principal (archivos de cabecera) para C y C++ tienen la extensión de archivo `.h`. Hay 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 y Linux, el precompilador C++ busca `payroll.sqC`, luego `payroll.sqx`, luego `payroll.hpp` y luego `payroll.h` en cada uno de los directorios que examina. En los sistemas operativos Windows de 32 bits, el precompilador C++ busca sucesivamente `payroll.sqx`, `payroll.hpp` y `payroll.h` en cada uno de los directorios que examina.

- `EXEC SQL INCLUDE 'pay/payroll.h';`

  Si el nombre de archivo está encerrado entre comillas, como en el caso anterior, no se añade ninguna extensión al nombre.

  Si el nombre de archivo entrecomillado no contiene una vía de acceso absoluta, se utiliza el contenido de `DB2INCLUDE` para buscar el archivo, y se le añade como prefijo la vía de acceso especificada en el nombre de archivo de `INCLUDE`. Por ejemplo, en los sistemas basados en UNIX y Linux, si `DB2INCLUDE` se establece en '`/disk2:myfiles/c`', el precompilador C o 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 la que se encuentra realmente el archivo. En sistemas operativos 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 #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 del número de línea del archivo fuente de salida precompilado.

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.

A continuación se describen los archivos de inclusión destinados a su uso en las aplicaciones del usuario.

SQLADEF (sqladef.h)
: Este archivo contiene prototipos de funciones utilizados por aplicaciones C y C++ precompiladas.

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.

SQLCODES (sqlcodes.h)
: Este archivo define constantes para el campo SQLCODE de la estructura SQLCA.

SQLDA (sqlda.h)
: Este archivo define la estructura del Área de descriptores de SQL (SQLDA). La SQLDA se utiliza para pasar datos entre una aplicación y el gestor de bases de datos.

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.

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).

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.

Archivos de inclusión para aplicaciones de SQL incorporado COBOL

Los archivos de inclusión 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 de inclusión 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 de inclusión 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 de inclusión de DB2 para las aplicaciones se encuentran en el directorio siguiente:

```
$HOME/sqllib/include/cobol_a
```

Para localizar archivos 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 sistemas de bases de datos 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 valor de DB2INCLUDE se coloca en la antememoria mediante procesador de línea de mandatos de DB2. 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.

A continuación se describen los archivos de inclusión destinados a su uso en las aplicaciones del usuario.

SQLCA (sqlca.cbl)

Este archivo define la estructura del Área de comunicaciones de SQL (SQLCA). El SQLCA contiene variables que utiliza el gestor de bases de datos para proporcionar a una aplicación información de error sobre la ejecución de sentencias de SQL y llamadas a API.

SQLCA_92 (sqlca_92.cbl)
Este archivo contiene una versión que se ajusta a FIPS SQL92 Entry Level de la estructura Área de comunicaciones de SQL (SQL Communications Area (SQLCA)). Debe incluir este archivo en lugar del archivo `sqlca.cbl` cuando escriba aplicaciones DB2 que se ajusten al estándar FIPS SQL92 Entry Level. El precompilador de DB2 incluye automáticamente el archivo `sqlca_92.cbl` cuando la opción LANGLEVEL del precompilador se establece en SQL92E.

SQLCODES (sqlcodes.cbl)
Este archivo define constantes para el campo SQLCODE de la estructura SQLCA.

SQLDA (sqlda.cbl)
Este archivo define la estructura del Área de descriptores de SQL (SQLDA). La SQLDA se utiliza para pasar datos entre una aplicación y el gestor de bases de datos.

SQLEAU (sqleau.cbl)
Este archivo contiene definiciones de constantes y de estructuras necesarias para las API de auditoría de seguridad de DB2. Si utiliza estas API, tiene que incluir este archivo en el programa. Este archivo también contiene definiciones de constantes y de valores de palabras clave para los campos del registro de seguimiento de auditoría. Programas externos o de extracción de seguimiento de auditoría de proveedores pueden utilizar estas definiciones.

SQLETSD (sqletsd.cbl)
Este archivo define la estructura Descriptor de espacio de tabla (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.

SQLSTATE (sqlstate.cbl)
Este archivo define constantes correspondientes al campo SQLSTATE de la estructura SQLCA.

SQLUDF (sqludf.cbl)
Este archivo define constantes y estructuras de interfaces para escribir funciones definidas por el usuario (UDF).

SQLUTBCQ (sqlutbcq.cbl)
Este archivo define la estructura de datos (Consulta de contenedor de espacio de tabla (Table Space Container Query), SQLB-TBSCONTQRY-DATA, que se utiliza con las API de consulta del contenedor del espacio de tabla, sqlgstsc, sqlgftcq y sqlgtcq.

SQLUTBSQ (sqlutbsq.cbl)
Este archivo define la estructura de datos Consulta de espacio de tabla (Table Space Query), SQLB-TBSTQRY-DATA, que se utiliza con las API de consulta del espacio de tabla, sqlgstsq, sqlgftsq y sqlgtsq.

Archivos de inclusión para aplicaciones de SQL incorporado FORTRAN

Los archivos de inclusión específicos del lenguaje principal correspondientes a FORTRAN tienen la extensión de archivo `.f` en los sistemas operativos UNIX y Linux y `.for` en los sistemas operativos Windows. Existen dos métodos para incluir archivos: la sentencia EXEC SQL INCLUDE y la sentencia INCLUDE de FORTRAN. El precompilador pasará por alto las sentencias INCLUDE de FORTRAN y sólo procesará los archivos incluidos mediante la sentencia EXEC SQL. Para localizar el archivo INCLUDE, el precompilador FORTRAN de DB2 busca en primer lugar en el directorio actual y, a continuación, en los directorios especificados por la variable de entorno DB2INCLUDE.

Considere los ejemplos siguientes:

- `EXEC SQL INCLUDE payroll`

Si el archivo especificado en la sentencia INCLUDE no está encerrado entre comillas, como en el ejemplo anterior, el precompilador busca sucesivamente `payroll.sqf` y `payroll.f` (`payroll.for` en sistemas operativos Windows) en cada uno de los directorios que examina.

- `EXEC SQL INCLUDE 'pay/payroll.f'`

  Si el nombre de archivo está encerrado entre comillas, como en el caso anterior, no se añade ninguna extensión al nombre. (Para sistemas operativos 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, y se le añade como prefijo la vía de acceso especificada en el nombre de archivo de INCLUDE. Por ejemplo, en DB2 para los sistemas operativos UNIX y Linux, si DB2INCLUDE tiene el valor '`/disk2:myfiles/fortran`', el precompilador busca sucesivamente '`./pay/payroll.f`', '`/disk2/pay/payroll.f`' y finalmente '`./myfiles/cobol/pay/payroll.f`'. En los mensajes del precompilador se muestra la vía de acceso en la que se encuentra realmente el archivo. En los sistemas operativos Windows, sustituya las barras inclinadas invertidas (\) por barras inclinadas y sustituya `'for'` por la extensión `'f'` en el ejemplo anterior.

Nota: El valor de DB2INCLUDE se coloca en la antememoria mediante procesador de línea de mandatos de DB2. 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.

Los archivos de cabecera FORTRAN de 32 bits necesarios para el desarrollo de aplicaciones de bases de datos DB2, que antes se encontraban en `$INSTHOME/sqllib/include` ahora se encuentran en `$INSTHOME/sqllib/include32`.

En la Versión 8.1, estos archivos se encontraban en el directorio `$INSTDIR/sqllib/include`, que era un enlace simbólico a uno de los directorios siguientes: `$DB2DIR/include` o `$DB2DIR/include64` según si se trataba de una instancia de 32 bits o de 64 bits.

En la Versión 9.1, `$DB2DIR/include` contendrá todos los archivos de inclusión (de 32 bits y de 64 bits) y `$DB2DIR/include32` sólo contendrá los archivos FORTRAN de 32 bits y un archivo README que indicará que los archivos de inclusión de 32 bits son los mismos que los de 64 bits con la excepción de FORTRAN.

El directorio `$DB2DIR/include32` sólo existirá en AIX, Solaris, HP-PA y HP-IPF.

Puede utilizar los siguientes archivos de inclusión de FORTRAN en las aplicaciones.

SQLCA (sqlca_cn.f, sqlca_cs.f)
: Este archivo define la estructura del Área de comunicaciones de SQL (SQLCA). El SQLCA contiene variables que utiliza el gestor de bases de datos para proporcionar a una aplicación información de error sobre la ejecución de sentencias de SQL y llamadas a API.

  Se proporcionan dos archivos SQLCA para las aplicaciones FORTRAN. El valor por omisión, `sqlca_cs.f`, define la estructura SQLCA en un formato compatible con SQL de IBM. El archivo `sqlca_cn.f`, precompilado con la opción `SQLCA NONE`, define la estructura SQLCA para un mejor rendimiento.

SQLCA_92 (sqlca_92.f)
: Este archivo contiene una versión que se ajusta a FIPS SQL92 Entry Level de la estructura Área de comunicaciones de SQL (SQL Communications

Area (SQLCA)). Se debe incluir este archivo en lugar de los archivos `sqlca_cn.f` o `sqlca_cs.f` cuando se escriban aplicaciones DB2 que se ajusten al estándar FIPS SQL92 Entry Level. El precompilador de DB2 incluye automáticamente el archivo `sqlca_92.f` cuando la opción LANGLEVEL del precompilador se establece en SQL92E.

SQLCODES (sqlcodes.f)
Este archivo define constantes para el campo SQLCODE de la estructura SQLCA.

SQLDA (sqldact.f)
Este archivo define la estructura del Área de descriptores de SQL (SQLDA). La SQLDA se utiliza para pasar datos entre una aplicación y el gestor de bases de datos.

SQLEAU (sqleau.f)
Este archivo contiene definiciones de constantes y de estructuras necesarias para las API de auditoría de seguridad de DB2. Si utiliza estas API, tiene que incluir este archivo en el programa. Este archivo también contiene definiciones de constantes y de valores de palabras clave para los campos del registro de seguimiento de auditoría. Programas externos o de extracción de seguimiento de auditoría de proveedores pueden utilizar estas definiciones.

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.

SQLSTATE (sqlstate.f)
Este archivo define constantes correspondientes al campo SQLSTATE de la estructura SQLCA.

SQLUDF (sqludf.f)
Este archivo define constantes y estructuras de interfaces para escribir funciones definidas por el usuario (UDF).

Declaración de la SQLCA para el manejo de errores

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

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

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

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

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

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

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`.

Manejo de errores utilizando la sentencia WHENEVER

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

La sentencia WHENEVER tiene tres formatos básicos:

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

En las sentencias anteriores:

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

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

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

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

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

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

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

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

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

Conexión con bases de datos DB2 en aplicaciones de SQL incorporado

Para poder trabajar con una base de datos, tiene que establecer una conexión con dicha base de datos. SQL incorporado proporciona varias formas que le permiten incluir código para establecer conexiones de base de datos. En función del lenguaje de programación principal de SQL incorporado, puede haber una o varias formas de hacerlo.

Las conexiones de base de datos se pueden establecer de forma implícita o explícita. Una conexión implícita es una conexión en la que se supone que el ID de usuario es el ID de usuario actual. Este tipo de conexión no se recomienda para aplicaciones de bases de datos. Se recomienda utilizar conexiones de bases de datos explícitas, que necesitan que se especifique un ID de usuario y una contraseña.

Conexión con bases de datos DB2 en aplicaciones de SQL incorporado C y C++

Cuando se trabaja con aplicaciones C y C++, se puede establecer una conexión de base de datos ejecutando la siguiente sentencia.

```
EXEC SQL CONNECT TO sample;
```

Si desea utilizar un id de usuario (`herrick`) y una contraseña (`mypassword`) específicos, utilice la siguiente sentencia:

```
EXEC SQL CONNECT TO sample USER herrick USING mypassword;
```

Conexión con bases de datos DB2 en aplicaciones de SQL incorporado COBOL

Cuando se trabaja con aplicaciones COBOL, se establece una conexión de base de datos ejecutando la siguiente sentencia. Esta sentencia crea una conexión con la base de datos sample utilizando el nombre de usuario por omisión.

```
EXEC SQL CONNECT TO sample END-EXEC.
```

Si desea utilizar un id de usuario (`herrick`) y una contraseña (`mypassword`) específicos, utilice la siguiente sentencia:

```
EXEC SQL CONNECT TO sample USER herrick USING mypassword END-EXEC.
```

Conexión con bases de datos DB2 en aplicaciones de SQL incorporado FORTRAN

Cuando se trabaja con aplicaciones FORTRAN, se establece una conexión de base de datos ejecutando la siguiente sentencia. Esta sentencia crea una conexión con la base de datos sample utilizando el nombre de usuario por omisión.

```
EXEC SQL CONNECT TO sample
```

Si desea utilizar un id de usuario (`herrick`) y una contraseña (`mypassword`) específicos, utilice la siguiente sentencia:

```
EXEC SQL CONNECT TO sample USER herrick USING mypassword
```

Conexión con bases de datos DB2 en aplicaciones de SQL incorporado REXX

Cuando se trabaja con aplicaciones REXX, se establece una conexión de base de datos ejecutando la siguiente sentencia. Esta sentencia crea una conexión con la base de datos sample utilizando el nombre de usuario por omisión.

```
CALL SQLEXEC 'CONNECT TO sample'
```

Si desea utilizar un id de usuario (`herrick`) y una contraseña (`mypassword`) específicos, utilice la siguiente sentencia:

```
CALL SQLEXEC 'CONNECT TO sample USER herrick USING mypassword'
```

Tipos de datos que se correlacionan con tipos de datos SQL en aplicaciones de SQL incorporado

Para intercambiar datos entre una aplicación y la base de datos, utilice las correlaciones de tipos de datos correctas para las variables utilizadas. Cuando el precompilador encuentra una declaración de una variable del lenguaje principal, determina el valor del tipo de SQL apropiado. Con cada lenguaje principal hay reglas de correlación especiales a las que hay que ceñirse, exclusivas para ese lenguaje específico.

Tipos de datos de SQL soportados en aplicaciones de SQL incorporado C y C++

Ciertos tipos de datos predefinidos de C y C++ corresponden a los tipos de columna de base de datos de DB2. Sólo se pueden declarar como variables del lenguaje principal estos tipos de datos de C y C++.

Las tablas siguientes muestran el equivalente en C y 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.

Tabla 2. Tipos de datos de SQL correlacionados con declaraciones de C y C++

Tipo de columna SQL[1]	Tipo de datos C y C++	Descripción del tipo de columna de 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

Tabla 2. Tipos de datos de SQL correlacionados con declaraciones de C y C++ (continuación)

Tipo de columna SQL[1]	Tipo de datos C y C++	Descripción del tipo de columna de SQL
DOUBLE[5] (480 ó 481)	double	Coma flotante de precisión doble
DECIMAL(*p*, *s*) (484 ó 485)	No existe un equivalente exacto; utilice double	Decimal empaquetado (Considere la posibilidad de utilizar las funciones CHAR y DECIMAL para manipular los campos decimales empaquetados como datos de tipo carácter.)
CHAR(1) (452 ó 453)	char	Carácter simple
CHAR(*n*) (452 ó 453)	No existe un equivalente exacto; utilice char[*n+1*], donde n es suficientemente grande para contener los datos 1<=*n*<=254	Serie de caracteres de longitud fija

VARCHAR(*n*)
(448 ó 449)
	struct tag {
	short int;
	char[*n*]
	}
	
	1<=*n*<=32 672
		Serie de caracteres de longitud variable no terminada en nulo con indicador de longitud de serie de 2 bytes

	Como alternativa, utilice char[*n+1*], donde n es suficientemente grande para contener los datos 1<=*n*<=32 672	Serie de caracteres de longitud variable terminada en nulo **Nota:** Se le asigna un tipo SQL de 460/461.

LONG VARCHAR
(456 ó 457)
	struct tag {
	short int;
	char[*n*]
	}
	
	32 673<=*n*<=32 700
		Serie de caracteres de longitud variable no terminada en nulo con indicador de longitud de serie de 2 bytes

CLOB(*n*) (408 ó 409)	sql type is clob(*n*) 1<=*n*<=2 147 483 647	Serie de caracteres de longitud variable no terminada en nulo con indicador de longitud de serie de 4 bytes
Variable de localizador CLOB[6] (964 ó 965)	sql type is clob_locator	Identifica las entidades CLOB que residen en el servidor
Variable de referencia de archivo CLOB[6] (920 ó 921)	sql type is clob_file	Descriptor del archivo que contiene datos CLOB

Tabla 2. Tipos de datos de SQL correlacionados con declaraciones de C y C++ (continuación)

Tipo de columna SQL[1]	Tipo de datos C y C++	Descripción del tipo de columna de SQL
BLOB(*n*) (404 ó 405)	sql type is blob(*n*) 1<=*n*<=2 147 483 647	Serie binaria variable no terminada en nulo con indicador de longitud de serie de 4 bytes
Variable de localizador BLOB[6] (960 ó 961)	sql type is blob_locator	Identifica las entidades BLOB del servidor
Variable de referencia de archivo BLOB[6] (916 ó 917)	sql type is blob_file	Descriptor del archivo que contiene datos BLOB
DATE (384 ó 385)	Formato de caracteres terminado en nulo	Admitir como mínimo 11 caracteres para acomodar el terminador nulo
	Formato estructurado VARCHAR	Admitir como mínimo 10 caracteres
TIME (388 ó 389)	Formato de caracteres terminado en nulo	Admitir como mínimo 9 caracteres para acomodar el terminador nulo
	Formato estructurado VARCHAR	Admitir como mínimo 8 caracteres
TIMESTAMP (392 ó 393)	Formato de caracteres terminado en nulo	Admitir como mínimo 27 caracteres para acomodar el terminador nulo
	Formato estructurado VARCHAR	Admitir como mínimo 26 caracteres

XML[7]
(988 ó 989)
	struct {
	sqluint32 length;
	char data[*n*];
	}
	
	1<=*n*<=2 147 483 647
	
	SQLUDF_CLOB
		Valor XML

BINARY	unsigned char myBinField[4]; 1<= *n* <=255	Datos binarios
VARBINARY	struct myVarBinField_t {sqluint16 length;char data[12];} myVarBinField; 1<= *n* <=32704	Datos varbinary

Los tipos de datos siguientes sólo están disponibles en los entornos DBCS o EUC si se compilan previamente con la opción WCHARTYPE NOCONVERT.

Tabla 3. Tipos de datos de SQL correlacionados con declaraciones de C y C++

Tipo de columna SQL[1]	Tipo de datos C y C++	Descripción del tipo de columna de SQL
GRAPHIC(1) (468 ó 469)	sqldbchar	Carácter simple de doble byte

Tabla 3. Tipos de datos de SQL correlacionados con declaraciones de C y C++ (continuación)

Tipo de columna SQL[1]	Tipo de datos C y C++	Descripción del tipo de columna de SQL
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

Los tipos de datos siguientes sólo están disponibles en los entornos DBCS o EUC si se compilan previamente con la opción WCHARTYPE CONVERT.

Tabla 4. Tipos de datos de SQL correlacionados con declaraciones de C y C++

Tipo de columna SQL[1]	Tipo de datos C y C++	Descripción del tipo de columna de SQL
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 4. Tipos de datos de SQL correlacionados con declaraciones de C y C++ (continuación)

Tipo de columna SQL[1]	Tipo de datos C y C++	Descripción del tipo de columna de 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

Los tipos de datos siguientes sólo están disponibles en los entornos DBCS o EUC.

Tabla 5. Tipos de datos de SQL correlacionados con declaraciones de C y C++

Tipo de columna SQL[1]	Tipo de datos C y C++	Descripción del tipo de columna de SQL
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
Referencia a archivos DBCLOB DBCLOB[6] (924 ó 925)	sql type is dbclob_file	Descriptor del archivo que contiene datos DBCLOB

Tabla 5. Tipos de datos de SQL correlacionados con declaraciones de C y C++ (continuación)

Tipo de columna SQL[1]	Tipo de datos C y C++	Descripción del tipo de columna de SQL

Nota:

1. El primer número debajo de **Tipo de columna de SQL** indica que no se proporciona una variable de indicador, y el segundo número indica que sí se proporciona dicha variable. Se necesita una variable de indicador para indicar los valores nulos (NULL) o para contener la longitud de una serie truncada. Éstos son los valores que aparecerán en el campo SQLTYPE de la SQLDA para estos tipos de datos.
2. Por razones de compatibilidad entre plataformas, utilice sqlint32. En los sistemas operativos UNIX y Linux de 64 bits, "long" es un entero de 64 bits. En los sistemas operativos Windows de 64 bits y UNIX y Linux 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 del sistema de bases de datos DB2 definirá el tipo de sqlint64 como "__int64" en los sistemas operativos Windows soportados cuando se utilice el compilador de Microsoft, como "long long" en los sistemas operativos UNIX y Linux de 32 bits y como "long" en los sistemas operativos UNIX y Linux de 64 bits.
4. FLOAT(*n*) donde $0 < n < 25$ es un sinónimo de REAL. La diferencia entre REAL y DOUBLE en la SQLDA es el valor de la longitud (4 u 8).
5. Los tipos de SQL siguientes son sinónimos de DOUBLE:
   - FLOAT
   - FLOAT(*n*) donde $24 < n < 54$ es un sinónimo de DOUBLE
   - DOUBLE PRECISION
6. Éste no es un tipo de columna, sino un tipo de variable del lenguaje principal.
7. El valor SQL_TYP_XML/SQL_TYP_NXML sólo se devuelve por parte de peticiones DESCRIBE. No se puede utilizar directamente por parte de la aplicación para vincular recursos de aplicaciones con valores XML.

A continuación mostramos normas adicionales para los tipos de datos C y 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 bajo 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 y C++ no está permitido, puesto que su representación interna depende de la máquina.

Tipos de datos para procedimientos, funciones y métodos en aplicaciones de SQL incorporado C y C++

La tabla siguiente lista las correlaciones soportadas entre tipos de datos de SQL y tipos de datos de C y C++ para procedimientos, UDF y métodos.

Tabla 6. Tipos de datos de SQL correlacionados con declaraciones de C y C++

Tipo de columna SQL[1]	Tipo de datos C y C++	Descripción del tipo de columna de SQL
SMALLINT (500 ó 501)	short	Entero con signo de 16 bits
INTEGER (496 ó 497)	sqlint32	Entero con signo de 32 bits
BIGINT (492 ó 493)	sqlint64	Entero con signo de 64 bits
REAL (480 ó 481)	float	Coma flotante de precisión simple
DOUBLE (480 ó 481)	double	Coma flotante de precisión doble
DECIMAL(*p*, *s*) (484 ó 485)	No soportado	Para pasar un valor decimal, defina el parámetro como tipo de datos convertible desde DECIMAL (por ejemplo, CHAR o DOUBLE) y convierta explícitamente el argumento a este tipo.
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

Tabla 6. Tipos de datos de SQL correlacionados con declaraciones de C y C++ (continuación)

Tipo de columna SQL[1]	Tipo de datos C y C++	Descripción del tipo de columna de SQL

LONG VARCHAR
(456 ó 457)
	struct {
	sqluint16 length;
	char[*n*]
	}
	
	32 673<=*n*<=32 700
		Serie de caracteres de longitud variable no terminada en nulo

CLOB(*n*)
(408 ó 409)
	struct {
	sqluint32 length;
	char data[n];
	}
	
	1<=*n*<=2 147 483 647
		Serie de caracteres de longitud variable no terminada en nulo con indicador de longitud de serie de 4 bytes

BLOB(*n*)
(404 ó 405)
	struct {
	sqluint32 length;
	char data[n];
	}
	
	1<=*n*<=2 147 483 647
		Serie binaria de longitud variable no terminada en nulo con indicador de longitud de serie de 4 bytes

DATE (384 ó 385)	char[11]	Formato de caracteres terminado en nulo
TIME (388 ó 389)	char[9]	Formato de caracteres terminado en nulo
TIMESTAMP (392 ó 393)	char[27]	Formato de caracteres terminado en nulo
XML (988/989)	No soportado	Este valor de tipo de descriptor (988/989) se definirá para que se utilice en la SQLDA para describir y para indicar datos XML (en su formato serializado). Los tipos de caracteres y binarios existentes (incluidos los LOB y los tipos de referencia de archivos LOB) también se pueden utilizar para captar e insertar los datos (sólo SQL dinámico)

Nota: Los tipos de datos siguientes sólo están disponibles en los entornos DBCS o EUC si se compilan previamente con la opción WCHARTYPE NOCONVERT.

Tabla 7. Tipos de datos de SQL correlacionados con declaraciones de C y C++

Tipo de columna SQL[1]	Tipo de datos C y C++	Descripción del tipo de columna de SQL
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

Tabla 7. Tipos de datos de SQL correlacionados con declaraciones de C y C++ (continuación)

Tipo de columna SQL[1]	Tipo de datos C y C++	Descripción del tipo de columna de SQL
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

Nota:

1. El primer número debajo de **Tipo de columna de SQL** indica que no se proporciona una variable de indicador, y el segundo número indica que sí se proporciona dicha variable. Se necesita una variable de indicador para indicar los valores nulos (NULL) o para contener la longitud de una serie truncada. Éstos son los valores que aparecerán en el campo SQLTYPE de la SQLDA para estos tipos de datos.

Tipos de datos de SQL soportados en aplicaciones de SQL incorporado COBOL

Ciertos tipos de datos de COBOL predefinidos corresponden a tipos de columna de base de datos DB2. Sólo estos tipos de datos de COBOL se pueden declarar como variables del lenguaje principal.

La tabla siguiente muestra el equivalente en COBOL de cada tipo de columna. Cuando el precompilador encuentra una declaración de 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 reconoce cada descripción de datos posible para las variables del lenguaje principal. Los elementos de datos de COBOL deben ser coherentes con los descritos en la tabla siguiente. Si utiliza otros elementos de datos, se puede producir un error.

Tabla 8. Tipos de datos de SQL correlacionados con declaraciones de COBOL

Tipo de columna SQL[1]	Tipo de datos de COBOL	Descripción del tipo de columna de SQL
SMALLINT (500 ó 501)	01 name PIC S9(4) COMP-5.	Entero con signo de 16 bits

Tabla 8. Tipos de datos de SQL correlacionados con declaraciones de COBOL (continuación)

Tipo de columna SQL[1]	Tipo de datos de COBOL	Descripción del tipo de columna de SQL
INTEGER (496 ó 497)	01 name PIC S9(9) COMP-5.	Entero con signo de 32 bits
BIGINT (492 ó 493)	01 name PIC S9(18) COMP-5.	Entero con signo de 64 bits
DECIMAL(*p, s*) (484 ó 485)	01 name PIC S9(*m*)V9(*n*) COMP-3.	Decimal empaquetado
REAL[2] (480 ó 481)	01 name USAGE IS COMP-1.	Coma flotante de precisión simple
DOUBLE[3] (480 ó 481)	01 name USAGE IS COMP-2.	Coma flotante de precisión doble
CHAR(*n*) (452 ó 453)	01 name PIC X(*n*).	Serie de caracteres de longitud fija
VARCHAR(*n*) (448 ó 449)	01 name. 49 length PIC S9(4) COMP-5. 49 name PIC X(*n*). 1<=*n*<=32 672	Serie de caracteres de longitud variable
LONG VARCHAR (456 ó 457)	01 name. 49 length PIC S9(4) COMP-5. 49 data PIC X(*n*). 32 673<=*n*<=32 700	Serie de caracteres de longitud variable larga
CLOB(*n*) (408 ó 409)	01 MY-CLOB USAGE IS SQL TYPE IS CLOB(n). 1<=*n*<=2 147 483 647	Serie de caracteres de longitud variable y objeto grande
Variable de localizador 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 o 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 entidades BLOB que residen en el servidor

Tabla 8. Tipos de datos de SQL correlacionados con declaraciones de COBOL (continuación)

Tipo de columna SQL[1]	Tipo de datos de COBOL	Descripción del tipo de columna de SQL
Variable de referencia de archivo BLOB[4] (916 ó 917)	01 MY-BLOB-FILE USAGE IS SQL TYPE IS BLOB-FILE.	Descriptor del archivo que contiene datos BLOB
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
XML[5] (988 ó 989)	01 name USAGE IS SQL TYPE IS XML AS CLOB (size).	Valor XML

Los tipos de datos siguientes sólo están disponibles en el entorno DBCS.

Tabla 9. Tipos de datos de SQL correlacionados con declaraciones de COBOL

Tipo de columna SQL[1]	Tipo de datos de COBOL	Descripción del tipo de columna de SQL
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 de longitud variable con indicador de longitud de serie de 2 bytes
LONG VARGRAPHIC (472 ó 473)	01 name. 49 length PIC S9(4) COMP-5. 49 name PIC G(*n*) DISPLAY-1. 16 337<=*n*<=16 350	Serie de caracteres de doble byte de longitud variable con indicador de longitud de serie de 2 bytes
DBCLOB(*n*) (412 ó 413)	01 MY-DBCLOB USAGE IS SQL TYPE IS DBCLOB(n). 1<=*n*<=1 073 741 823	Serie de caracteres de doble byte de longitud variable y de objeto grande con indicador de longitud de serie de 4 bytes
Variable de localizador 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 archivo DBCLOB[4] (924 ó 925)	01 MY-DBCLOB-FILE USAGE IS SQL TYPE IS DBCLOB-FILE.	Descriptor de archivo que contiene datos DBCLOB

Tabla 9. Tipos de datos de SQL correlacionados con declaraciones de COBOL (continuación)

Tipo de columna SQL[1]	Tipo de datos de COBOL	Descripción del tipo de columna de SQL

Nota:

1. El primer número debajo de **Tipo de columna de SQL** indica que no se proporciona una variable de indicador, y el segundo número indica que sí se proporciona dicha variable. Se necesita una variable de indicador para indicar los valores nulos (NULL) o para contener la longitud de una serie truncada. Éstos son los valores que aparecerán en el campo SQLTYPE de la SQLDA para estos tipos de datos.
2. FLOAT(*n*) donde 0 < *n* < 25 es un sinónimo de REAL. La diferencia entre REAL y DOUBLE en la SQLDA es el valor de la longitud (4 u 8).
3. Los tipos de SQL siguientes son sinónimos de DOUBLE:
   - FLOAT
   - FLOAT(*n*) donde 24 < *n* < 54 es sinónimo de DOUBLE.
   - DOUBLE PRECISION
4. Éste no es un tipo de columna, sino un tipo de variable del lenguaje principal.
5. El valor SQL_TYP_XML/SQL_TYP_NXML sólo se devuelve por parte de peticiones DESCRIBE. No se puede utilizar directamente por parte de la aplicación para vincular recursos de aplicaciones con valores XML.

A continuación, se facilitan reglas adicionales para los tipos de datos de COBOL soportados:

- PIC S9 y COMP-3/COMP-5 son necesarios donde se muestran.
- Se puede utilizar el número de nivel 77 en lugar del 01 para todos los tipos de columna, excepto VARCHAR, LONG VARCHAR, VARGRAPHIC, LONG VARGRAPHIC y todos los tipos de variable de LOB.
- Utilice las reglas siguientes al declarar variables del lenguaje principal para los tipos de columna DECIMAL(p,s). Vea el ejemplo siguiente:

  ```
  01 identifier PIC S9(m)V9(n) COMP-3
  ```

  – Utilice V para indicar la coma decimal.
  – Los valores para *n* y *m* deben ser superiores o equivalentes a 1.
  – El valor de *n* + *m* no puede ser superior a 31.
  – El valor para *s* equivale al valor para *n*.
  – El valor para *p* equivale al valor de *n* + *m*.
  – Los factores de repetición *(n)* y *(m)* son opcionales. Todos los ejemplos siguientes resultan válidos:

  ```
  01 identifier PIC S9(3)V COMP-3
  01 identifier PIC SV9(3) COMP-3
  01 identifier PIC S9V COMP-3
  01 identifier PIC SV9 COMP-3
  ```

  – Se puede utilizar PACKED-DECIMAL en lugar de COMP-3.
- Las matrices *no* están soportadas por el precompilador de COBOL.

Tipos de datos de SQL soportados en aplicaciones de SQL incorporado FORTRAN

Ciertos tipos de datos de FORTRAN predefinidos corresponden a tipos de columna de base de datos DB2. 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.

Tabla 10. Tipos de datos SQL correlacionados con declaraciones FORTRAN

Tipo de columna SQL[1]	Tipo de datos FORTRAN	Descripción del tipo de columna de 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* es 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 de 32 700	Serie de caracteres de longitud variable larga
CLOB(*n*) (408 ó 409)	SQL TYPE IS CLOB (n) donde *n* es 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 o 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* es 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

Tabla 10. Tipos de datos SQL correlacionados con declaraciones FORTRAN (continuación)

Tipo de columna SQL[1]	Tipo de datos FORTRAN	Descripción del tipo de columna de SQL
TIME (388 ó 389)	CHARACTER*8	Serie de caracteres de 8 bytes
TIMESTAMP (392 ó 393)	CHARACTER*26	Serie de caracteres de 26 bytes
XML (988 ó 989)	SQL_TYP_XML	No hay soporte de XML para FORTRAN; las aplicaciones pueden obtener el tipo de descripción, pero no pueden utilizarlo.

Nota:

1. El primer número debajo de **Tipo de columna de SQL** indica que no se proporciona una variable de indicador, y el segundo número indica que sí se proporciona dicha variable. Se necesita una variable de indicador para indicar los valores nulos (NULL) o para contener la longitud de una serie truncada. Éstos son los valores que aparecerán en el campo SQLTYPE de la SQLDA para estos tipos de datos.
2. FLOAT(*n*) donde $0 < n < 25$ es un sinónimo de REAL. La diferencia entre REAL y DOUBLE en la SQLDA es el valor de la longitud (4 u 8).
3. Los tipos de SQL siguientes son sinónimos de DOUBLE:
   - FLOAT
   - FLOAT(*n*) donde $24 < n < 54$ es sinónimo de DOUBLE.
   - 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.

Tipos de datos de SQL soportados en aplicaciones de SQL incorporado REXX

Ciertos tipos de datos de REXX predefinidos corresponden a tipos de columna de base de datos DB2. Sólo se pueden declarar como variables del lenguaje principal estos tipos de datos de REXX. La tabla siguiente muestra cómo SQLEXEC y SQLDBS interpretan variables de REXX para convertir su contenido en tipos de datos de DB2.

Tabla 11. 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 de SQL
SMALLINT (500 ó 501)	Un número sin coma decimal que va de -32 768 a 32 767	Entero con signo de 16 bits
INTEGER (496 ó 497)	Un número sin coma decimal que va de -2 147 483 648 a 2 147 483 647	Entero con signo de 32 bits
REAL[2] (480 ó 481)	Un número en notación científica que va de $-3.40282346 \times 10^{38}$ a $3.40282346 \times 10^{38}$	Coma flotante de precisión simple

Tabla 11. 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 de SQL
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
Referencia de archivo CLOB[4] (920 o 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

Tabla 11. 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 de SQL
TIMESTAMP (392 ó 393)	Equivalente a CHAR(*26*)	Serie de caracteres de 26 bytes
XML (988 ó 989)	SQL_TYP_XML	No hay soporte de XML para REXX; las aplicaciones pueden obtener el tipo de descripción, pero no pueden utilizarlo.

Los tipos de datos siguientes sólo están disponibles en el entorno DBCS.

Tabla 12. 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 de 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 fija 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 archivo DBCLOB[4] (924 ó 925)	DECLARE *:nombre_var* LANGUAGE TYPE DBCLOB FILE	Descriptor de archivo que contiene datos DBCLOB

Nota:

1. El primer número que se encuentra bajo **Tipo de columna SQL** indica que no se proporciona una variable de indicador, y el segundo número indica que se proporciona una variable de indicador. Se necesita una variable de indicador para indicar los valores nulos (NULL) o para contener la longitud de una serie truncada.
2. FLOAT(*n*) donde $0 < n < 25$ es un sinónimo de REAL. La diferencia entre REAL y DOUBLE en la SQLDA es el valor de la longitud (4 u 8).
3. Los tipos de SQL siguientes son sinónimos de DOUBLE:
   - FLOAT
   - FLOAT(*n*) donde $24 < n < 54$ es sinónimo de DOUBLE.
   - DOUBLE PRECISION
4. Éste no es un tipo de columna, sino un tipo de variable del lenguaje principal.

Variables del lenguaje principal en aplicaciones de SQL incorporado

Las *variables del lenguaje principal* son variables a las que hacen referencia las sentencias de SQL incorporado. Se utilizan para intercambiar valores de datos entre el servidor de bases de datos y la aplicación de SQL incorporado. Las aplicaciones de SQL incorporado también pueden incluir declaraciones de variables del lenguaje principal para consultas de SQL relacional. Además, se puede utilizar una variable del lenguaje principal para que contenga una expresión XQuery que se va a ejecutar. Sin embargo, no hay ningún mecanismo para pasar valores a parámetros en expresiones XQuery.

Las variables del lenguaje principal se declaran utilizando la sintaxis de declaración de variables específica del lenguaje principal en una sección de declaración.

Una sección de declaración es la parte de una aplicación de SQL incorporado que se encuentra junto a la parte superior de un archivo de código fuente de SQL incorporado y que se vincula mediante dos sentencias de SQL no ejecutables:

- BEGIN DECLARE SECTION
- END DECLARE SECTION

Estas sentencias permiten al precompilador encontrar las declaraciones de las variables. Cada declaración de variable del lenguaje principal debe aparecer entre estas dos sentencias; de lo contrario, las variables se consideran variables normales.

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, dentro de una sección de declaración bien formada, 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.
- Los nombres de variables del lenguaje principal deben ser exclusivos dentro de un archivo fuente. Esto se debe a que el precompilador de DB2 no es responsable de las normas de ámbito de variables específicas del lenguaje principal. Como tal, sólo hay un ámbito para las variables del lenguaje principal.

  Nota: 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;
  .
  .
  .
  }
```

Declaración de variables del lenguaje principal en aplicaciones de SQL incorporado

Para transmitir datos entre el servidor de bases de datos y la aplicación, tiene que declarar variables del lenguaje principal en el código fuente de la aplicación para cosas como consultas de SQL relacional y declaraciones de variables del lenguaje principal para expresiones de XQuery.

La tabla siguiente contiene ejemplos de declaraciones de variables del lenguaje principal para lenguajes principales de SQL incorporado.

Tabla 13. Declaraciones de variables del lenguaje principal por lenguaje principal

Lenguaje	Código fuente de ejemplo

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

Tabla 13. Declaraciones de variables del lenguaje principal por lenguaje principal (continuación)

Lenguaje	Código fuente de ejemplo

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).`
	`  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`
	`  integer*2  nul_ind`
	`EXEC SQL END DECLARE SECTION`

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.

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

```
db2dclgn -d nombre-basedatos -t nombre-tabla [opciones]
```

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

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

El archivo `staff.h` resultante contiene:

```
struct
{
  short id;
  struct
  {
    short length;
    char data[9];
  } name;
  short dept;
  char job[6];
  short   years;
  double    salary;
  double comm;
} staff;
```

Tipos de datos de columna y variables del lenguaje principal en aplicaciones de SQL incorporado

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.

Nota:

1. Cada tipo de datos soportado puede tener el atributo NOT NULL. Esto se trata como otro tipo.
2. Los 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 de SQL incorporado 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 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 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 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
```

La función `DIGITS` utilizada en el ejemplo anterior devuelve una representación de un número como una serie de caracteres.

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.

Declaración de variables del lenguaje principal XML en aplicaciones de SQL incorporado

Para intercambiar datos XML entre el servidor de bases de datos y una aplicación de SQL incorporado, tiene que declarar las variables del lenguaje principal en el código fuente de la aplicación.

DB2 V9.1 incorpora un tipo de datos XML que almacena los datos XML en un conjunto estructurado de nodos en un formato de árbol. Las columnas con este tipo de datos XML se describen como un SQLTYPE de columna SQL_TYP_XML y las aplicaciones pueden vincular varios tipos de datos específicos del lenguaje como entradas o salidas en estas columnas o parámetros. Se puede acceder directamente a las columnas XML mediante SQL, las extensiones de SQL/XML o XQuery. El tipo de datos XML se aplica a más elementos que columnas. Las funciones pueden tener argumentos de valores XML o generar valores XML. De forma similar, los procedimientos almacenados pueden tomar valores XML como parámetros de entrada y de salida. Finalmente, las expresiones de XQuery generan valores XML independientemente de si acceden a columnas XML o no.

Los datos XML son de tipo carácter por naturaleza y tienen una codificación que especifica el juego de caracteres utilizado. La codificación de los datos XML puede determinarse externamente, que se deriva del tipo de aplicación base que contiene la representación de serie serializada del documento XML. También puede determinarse internamente, que requiere la interpretación de los datos. Para los documentos codificados en Unicode, se recomienda utilizar un marcador de orden de bytes (BOM), consistente en un código de carácter Unicode al principio de una corriente de datos. El BOM se utiliza como una signatura que define el orden de los bytes y el formato de codificación Unicode.

Los tipos existentes de carácter y binarios, que incluyen CHAR, VARCHAR, CLOB y BLOB, se pueden utilizar además de las variables del lenguaje principal para captar e insertar datos. Sin embargo, no estarán sujetos al análisis de XML implícito, como lo estarían las variables del lenguaje principal XML. En su lugar, se inserta y se aplica una función XMLPARSE explícita con eliminación de espacios en blanco por omisión.

Restricciones de XML y XQuery al desarrollar aplicaciones de SQL incorporado

Para declarar variables del lenguaje principal XML en aplicaciones de SQL incorporado:

En la sección de declaración de la aplicación, declare las variables del lenguaje principal XML como tipos de datos LOB:

-
  ```
  SQL TYPE IS XML AS CLOB(n) <nombre_varpral>
  ```

  donde <nombre_varpral> es una variable del lenguaje principal CLOB que contiene datos XML codificados en la página de códigos mixta de la aplicación.

- `SQL TYPE IS XML AS DBCLOB(n) <nombre_varpral>`

  donde <nombre_varpral> es una variable del lenguaje principal DBCLOB que contiene datos XML codificados en la página de códigos gráfica de la aplicación.

- `SQL TYPE IS XML AS BLOB(n) <nombre_varpral>`

  donde <nombre_varpral> es una variable del lenguaje principal BLOB que contiene datos XML codificados internamente[1].

- `SQL TYPE IS XML AS CLOB_FILE <nombre_varpral>`

  donde <nombre_varpral> es un archivo CLOB que contiene datos XML codificados en la página de códigos mixta de la aplicación.

- `SQL TYPE IS XML AS DBCLOB_FILE <nombre_varpral>`

  donde <nombre_varpral> es un archivo DBCLOB que contiene datos XML codificados en la página de códigos gráfica de la aplicación.

- `SQL TYPE IS XML AS BLOB_FILE <nombre_varpral>`

  donde <nombre_varpral> es un archivo BLOB que contiene datos XML codificados internamente[1].

Nota:

1. Consulte el algoritmo para determinar la codificación con especificaciones XML 1.0 (`http://www.w3.org/TR/REC-xml/#sec-guessing-no-ext-info`).

Identificar valores XML en una SQLDA

Para indicar que un tipo base alberga datos XML, el campo sqlname de SQLVAR se debe actualizar del siguiente modo:

- `sqlname.length` debe ser 8
- Los dos primeros bytes de sqlname.data deben ser `X'0000'`
- El tercer y cuarto bytes de sqlname.data deben ser `X'0000'`
- El quinto byte de `sqlname.data` debe ser `X'01'` (que recibe el nombre del indicador de subtipo de XML sólo cuando se cumplen las dos primeras condiciones)
- Los restantes bytes deben ser `X'000000'`

Si el indicador de subtipo de XML está establecido en una SQLVAR cuyo SQLTYPE es no LOB, se devolverá un error SQL0804 (rc=115) en el momento de la ejecución.

Nota: SQL_TYP_XML sólo se puede devolver desde la sentencia DESCRIBE. Este tipo no se puede utilizar para ninguna otra petición. La aplicación debe modificar la SQLDA para que contenga un tipo binario o de carácter válido y debe establecer el campo `sqlname` de forma adecuada para que indique que los datos son XML.

Identificación de valores de SQL nulos con variables de indicador de nulo

Las aplicaciones de SQL incorporado deben prepararse para recibir valores nulos asociando una *variable de indicador de nulo* con cualquier variable del lenguaje principal que pueda recibir un nulo. El gestor de bases de datos y la aplicación del sistema principal comparten una variable de indicador nulo. Por lo tanto, esta variable se tiene que declarar en la aplicación como una variable del lenguaje principal, que corresponde al tipo de datos de SQL SMALLINT.

Una variable de indicador de nulo se coloca en una sentencia de SQL inmediatamente después de la variable del lenguaje principal y va precedida por un signo de dos puntos. Un espacio puede separar la variable de indicador de nulo 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 de nulo. También puede especificar una variable de indicador de nulo utilizando la palabra clave INDICATOR opcional, que debe colocar entre la variable del lenguaje principal y su indicador de nulo.

Se examina la variable de indicador de nulo para ver si contiene un valor negativo. Si el valor no es negativo, la aplicación puede utilizar el valor devuelto de la variable del lenguaje principal. Si el valor es negativo, el valor captado es nulo y la variable del lenguaje principal no se debe utilizar. 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 la variable de indicador de nulo puede ser -2. Este valor indica un nulo 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 nulos, la aplicación debe proporcionar un indicador de nulo. De lo contrario, puede producirse un error. Si no se utiliza un indicador de valor nulo, se devuelve un SQLCODE -305 (SQLSTATE 22002).

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

- Si la parte de segundos de un tipo de datos TIME está truncada, el valor del indicador de nulo 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 del indicador de nulo 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 de nulo, 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 nulo, si se permiten valores nulos.

Si la variable de indicador de nulo 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 de nulo.

El valor devuelto en la variable de indicador de nulo 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 nulo), 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 de nulo 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).

Para poder utilizar variables de indicador de nulo en el lenguaje principal, tiene que declarar las variables de indicador de nulo. En el siguiente ejemplo, programas C y C++ adecuados, la variable de indicador de nulo cmind se puede declarar del siguiente modo:

```
EXEC SQL BEGIN DECLARE SECTION;
  char cm[3];
  short cmind;
EXEC SQL END DECLARE SECTION;
```

La tabla siguiente contiene ejemplos correspondientes a los lenguajes principales soportados:

Tabla 14. Variables de indicador de nulo por lenguaje principal

Lenguaje	Código fuente de ejemplo

C y C++	`EXEC SQL FETCH C1 INTO :cm INDICATOR :cmind;`
	`if ( cmind < 0 )`
	`  printf( "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`

REXX	`  CALL SQLEXEC 'FETCH C1 INTO :cm INDICATOR :cmind'`
	`  IF ( cmind < 0 )`
	`  SAY 'Commission is NULL'`

Inclusión de las variables del lenguaje principal SQLSTATE y SQLCODE en aplicaciones de SQL incorporado

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. Si la aplicación cumple con el estándar FIPS 127-2, puede declarar las variables del lenguaje principal llamadas SQLSTATE y SQLCODE en lugar de declarar de forma explícita la estructura SQLCA en aplicaciones de SQL incorporado.

- Se tiene que especificar la opción de PREP LANGLEVEL SQL92E

En el siguiente ejemplo, la aplicación comprueba el campo SQLCODE de la estructura SQLCA para determinar si la actualización se ha realizado satisfactoriamente.

Tabla 15. Incorporación de sentencias de SQL en un lenguaje principal

Lenguaje	Código fuente de ejemplo

C y C++	`EXEC SQL UPDATE staff SET job = 'Clerk' WHERE job = 'Mgr';`
	`if ( SQLCODE < 0 )`
	`  printf( "Update Error:  SQLCODE =`

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`

Cómo hacer referencia a variables del lenguaje principal en aplicaciones de SQL incorporado

Cuando haya declarado una variable del lenguaje principal en el código de la aplicación de SQL incorporado, puede hacer referencia a la misma posteriormente en la 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 sintaxis de programación del lenguaje principal, omita el signo de dos puntos.

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

Tabla 16. Referencias a variables del lenguaje principal por lenguaje principal

Lenguaje	Código fuente de ejemplo
C o C++	`EXEC SQL FETCH C1 INTO :cm;` `printf( "Commission = %f\n", cm );`
COBOL	`EXEC SQL FETCH C1 INTO :cm END-EXEC` `DISPLAY 'Commission = ' cm`
FORTRAN	`EXEC SQL FETCH C1 INTO :cm` `WRITE(*,*) 'Commission = ', cm`
REXX	`CALL SQLEXEC 'FETCH C1 INTO :cm'` `SAY 'Commission = ' cm`

Ejemplo: cómo hacer referencia a variables del lenguaje principal XML en aplicaciones de SQL incorporado

Las siguientes aplicaciones de ejemplo hacen una demostración de cómo hacer referencia a las variable del lenguaje principal XML en C y en COBOL.

Ejemplo: aplicación C de SQL incorporado

```
El siguiente ejemplo de código se ha formateado para clarificarlo:
EXEC SQL BEGIN DECLARE;
  SQL TYPE IS XML AS CLOB( 10K ) xmlBuf;
  SQL TYPE IS XML AS BLOB( 10K ) xmlblob;
  SQL TYPE IS CLOB( 10K ) clobBuf;
EXEC SQL END DECLARE SECTION;

// como XML AS CLOB
// Al valor de XML escrito en xmlBuf se añadirá como prefijo una declaración
// de XML parecida a: <?xml version = "1.0" encoding = "ISO-8859-1" ?>
// Nota: el nombre de codificación dependerá de la página de códigos de la
// aplicación
EXEC SQL SELECT xmlCol INTO :xmlBuf
   FROM myTable
   WHERE id = '001';
EXEC SQL UPDATE myTable
   SET xmlCol = :xmlBuf
   WHERE id = '001';

// como XML AS BLOB
// Al valor de XML escrito en xmlblob se añadirá como prefijo una declaración
// de XML parecida a: <?xml version = "1.0" encoding = "UTF-8"?>
EXEC SQL SELECT xmlCol INTO :xmlblob
   FROM myTable
   WHERE id = '001';
EXEC SQL UPDATE myTable
   SET xmlCol = :xmlblob
   WHERE id = '001';

// como CLOB
// La salida estará codificada en la página de códigos de caracteres de la
// aplicación, pero nt contendrá una declaración XML
EXEC SQL SELECT XMLSERIALIZE (xmlCol AS CLOB(10K)) INTO :clobBuf
   FROM myTable
   WHERE id = '001';
EXEC SQL UPDATE myTable
   SET xmlCol = XMLPARSE (:clobBuf PRESERVE WHITESPACE)
   WHERE id = '001';
```

Ejemplo: aplicación COBOL de SQL incorporado

```
El siguiente ejemplo de código se ha formateado para clarificarlo:
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
  01 xmlBuf USAGE IS SQL TYPE IS XML as CLOB(5K).
  01 clobBuf USAGE IS SQL TYPE IS CLOB(5K).
  01 xmlblob  USAGE IS SQL TYPE IS BLOB(5K).
EXEC SQL END DECLARE SECTION END-EXEC.

* como XML
EXEC SQL SELECT xmlCol INTO :xmlBuf
   FROM myTable
   WHERE id = '001'.
EXEC SQL UPDATE myTable
   SET xmlCol = :xmlBuf
   WHERE id = '001'.

* como BLOB
EXEC SQL SELECT xmlCol INTO :xmlblob
   FROM myTable
```

```
   WHERE id = '001'.
EXEC SQL UPDATE myTable
   SET xmlCol = :xmlblob
   WHERE id = '001'.

* como CLOB
EXEC SQL SELECT XMLSERIALIZE(xmlCol AS CLOB(10K)) INTO :clobBuf
   FROM myTable
   WHERE id= '001'.
EXEC SQL UPDATE myTable
   SET xmlCol = XMLPARSE(:clobBuf) PRESERVE WHITESPACE
   WHERE id = '001'.
```

Variables del lenguaje principal en aplicaciones de SQL incorporado 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 a una aplicación intercambiar datos con el gestor de bases de datos. Una vez que se ha precompilado la aplicación, el compilador utiliza las variables del lenguaje principal como cualquier otra variable de C o C++. Cuando denomine, declare y utilice variables del lenguaje principal, siga las normas descritas en los apartados siguientes.

Consideraciones sobre variables de tipo long

En aplicaciones que construyen la SQLDA de forma manual, no se pueden utilizar variables tipo long cuando `sqlvar::sqltype==SQL_TYP_INTEGER`. En su lugar se deben utilizar los tipos `sqlint32`. Este problema es idéntico a utilizar variables long en declaraciones de variables del lenguaje principal, excepto en que con una SQLDA construida de forma manual el precompilador no revelará este error y se producirán errores en el tiempo de ejecución.

Las conversiones de long y unsigned long que se utilizan para acceder a información sqlvar::sqldata se deben cambiar por `sqlint32` y `sqluint32` respectivamente. Los miembros `Val` correspondientes a las estructuras `sqloptions` y `sqla_option` se declaran como `sqluintptr`. Por lo tanto, la asignación de miembros de puntero en miembros `sqla_option::val` o `sqloptions::val` deben utilizar conversiones `sqluintptr` en lugar de conversiones unsigned long. Este cambio no ocasionará problemas en el tiempo de ejecución en los sistemas operativos UNIX y Linux de 64 bits, pero se debe realizar como preparación para aplicaciones Windows de 64 bits, en las que el tipo long sólo es de 32 bits.

Consideraciones sobre la codificación de varios bytes

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). Las variables del lenguaje principal se elegirán consecuentemente, puesto que los datos gráficos de DB2 constan de 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 o 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.

Nombres de variables del lenguaje principal en aplicaciones de SQL incorporado C y C++

El precompilador SQL identifica las variables del lenguaje principal por el nombre declarado para éstas. Se aplican las siguientes normas:

- Los nombres de las variables del lenguaje principal no deben superar los 255 caracteres de longitud.
- Los nombres de las variables del lenguaje principal no deben utilizar el prefijo SQL, sql, DB2 ni db2, que 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.

Sección declare para variables del lenguaje principal en aplicaciones de SQL incorporado 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. Por ejemplo:

```
EXEC SQL BEGIN DECLARE SECTION;
  char  varsql;     /* permitido */
EXEC SQL END DECLARE SECTION;
```

El precompilador C o 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.

Ejemplo: Plantilla de sección de declaración de SQL para aplicaciones de SQL incorporado 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 */
      sql type is varbinary(12)
            myVarBinField;           /* SQL type  908 */
      sql type is binary(4)
            myBinField;              /* SQL type  912 */
      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;
```

Variables SQLSTATE y SQLCODE en una aplicación de SQL incorporado 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;
```

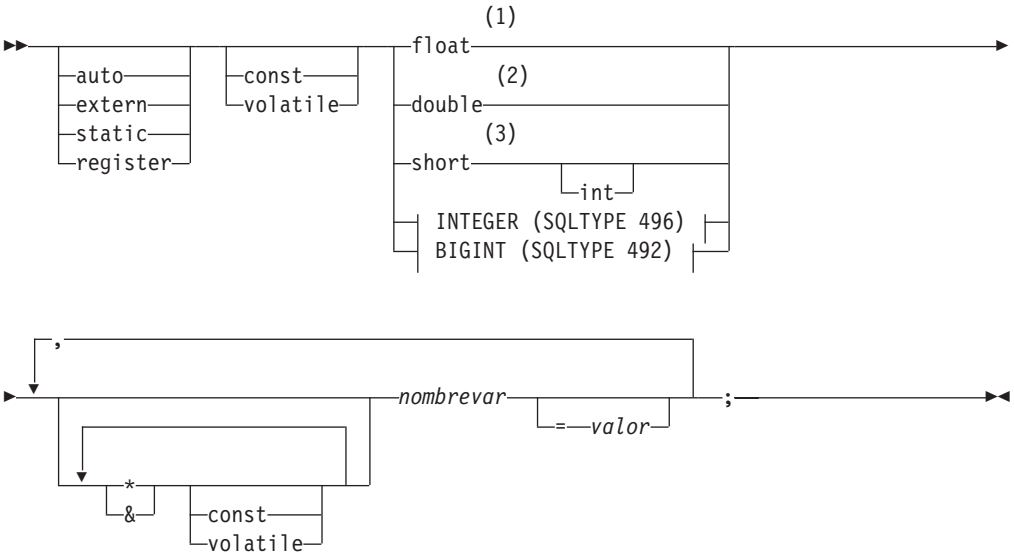
Se supone la declaración SQLCODE durante el paso de precompilación. Observe que, si se utiliza esta opción, no se debe especificar la sentencia INCLUDE SQLCA.

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];
```

Declaración de variables numéricas del lenguaje principal en aplicaciones de SQL incorporado C y C++

A continuación se muestra la sintaxis para declarar 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)

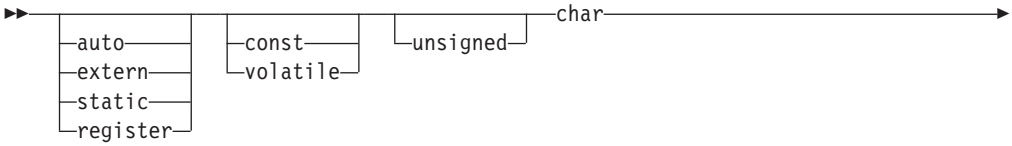
4 Para obtener una portabilidad máxima de las aplicaciones utilice, respectivamente, `sqlint32` y `sqlint64` para las variables del lenguaje principal INTEGER y BIGINT. Por omisión, el uso de variables del lenguaje principal tipo long tiene como consecuencia el error del precompilador SQL0402 en las plataformas en que la longitud es una cantidad de 64 bits, como por ejemplo en UNIX de 64 BITS. Utilice la opción LONGERROR NO de PREP para imponer que DB2 acepte variables long como tipos de variable del lenguaje principal aceptables y las trate como variables BIGINT.

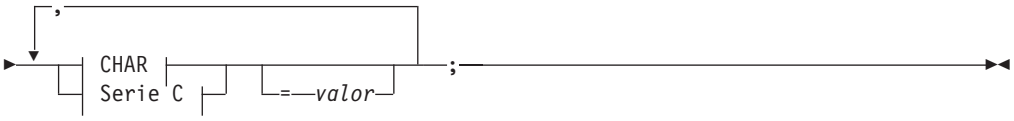
5 Para obtener una portabilidad máxima de las aplicaciones utilice, respectivamente, `sqlint32` y `sqlint64` para las variables del lenguaje principal INTEGER y BIGINT. Para utilizar el tipo de datos BIGINT, la plataforma tiene que soportar valores enteros de 64 bits. Por omisión, el uso de variables del lenguaje principal tipo long tiene como consecuencia el error del precompilador SQL0402 en las plataformas en que la longitud es una cantidad de 64 bits, como por ejemplo en UNIX de 64 BITS. Utilice la opción LONGERROR NO de PREP para imponer que DB2 acepte variables long como tipos de variable del lenguaje principal aceptables y las trate como variables BIGINT.

Declaración de variables del lenguaje principal de tipo carácter de longitud fija, terminadas en nulo y de longitud variable en aplicaciones de SQL incorporado C y C++

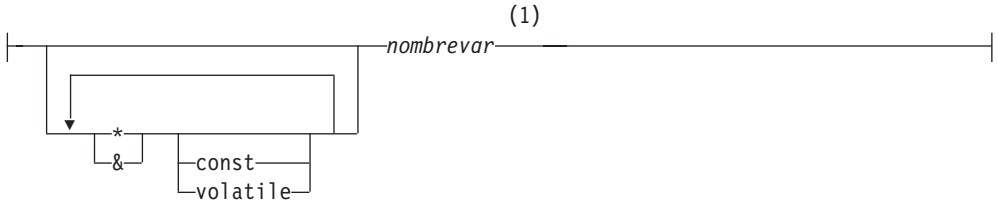
A continuación se muestra la sintaxis para declarar variables del lenguaje principal de tipo carácter fijas, terminadas en nulo (formato 1) y de longitud variable en C o C++ (formato 2).

Formato 1: Sintaxis correspondiente a variables del lenguaje principal de tipo carácter fijas y terminadas en nulo en aplicaciones de SQL incorporado en C o C++

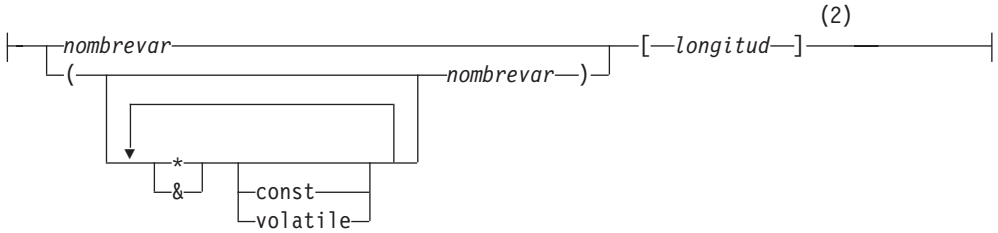




CHAR



Serie C

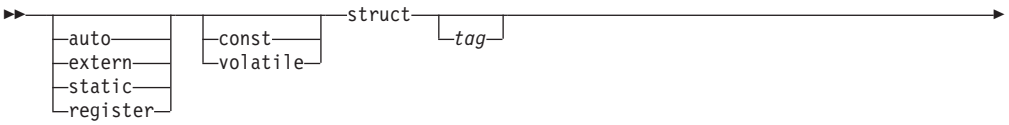


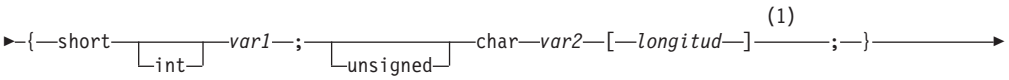
Notas:

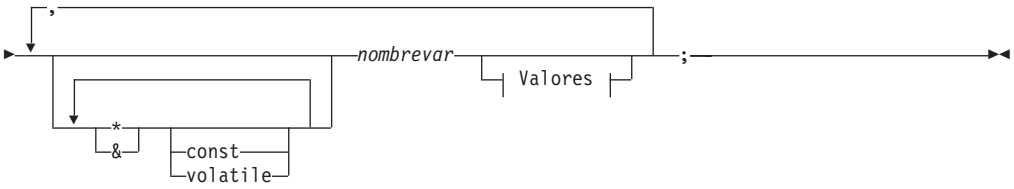
1 CHAR (SQLTYPE 452), length 1

2 Serie C terminada en nulo (SQLTYPE 460); la longitud puede ser cualquier expresión constante válida

Formato 2: Sintaxis correspondiente a variables del lenguaje principal de tipo carácter de longitud variable en aplicaciones de SQL incorporado C y 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.

Declaración de variables gráficas de lenguaje principal en aplicaciones de SQL incorporado en C y C++

Para manejar datos gráficos en aplicaciones C o C++, utilice variables de lenguaje principal basadas en el tipo de datos `wchar_t` de C o C++ o en el tipo de datos `sqldbchar` proporcionado por DB2. Puede asignar estos tipos de variables del lenguaje principal a las columnas de una tabla que sean GRAPHIC, VARGRAPHIC o DBCLOB. Por ejemplo, puede actualizar o seleccionar datos DBCS de las columnas GRAPHIC o VARGRAPHIC de una tabla.

Existen tres formatos válidos para una variable gráfica del lenguaje principal:

- Formato de gráfico simple

  Las variables de gráfico simple del lenguaje principal tienen para SQLTYPE el valor 468/469, lo que equivale al tipo de datos GRAPHIC(1) de SQL.
- Formato de gráfico terminado en nulo

  Terminado en nulo hace referencia a una situación en que todos los bytes del último carácter de la serie gráfica contienen ceros binarios ('\0's). Su tipo de datos de SQL es 400/401.
- Formato estructurado VARGRAPHIC

Las variables de formato estructurado VARGRAPHIC del lenguaje principal tienen para SQLTYPE el valor 464/465 si su longitud está comprendida entre 1 y 16.336 bytes. Tienen para SQLTYPE el valor 472/473 si su longitud está comprendida entre 2.000 y 16.350 bytes.

Tipos de datos wchar_t y sqldbchar para datos gráficos en aplicaciones de SQL incorporado 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 un sistema operativo Windows, tenga en cuenta que `wchar_t` en sistemas operativos Windows es Unicode. Por lo tanto, si el `wchar_t` del compilador C o 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).

Opción del precompilador WCHARTYPE para datos gráficos en aplicaciones de SQL incorporado C y C++

Mediante la opción WCHARTYPE del precompilador, puede especificar el formato de caracteres gráficos que desea utilizar en la aplicación C o 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 pueden 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`.

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.

Otras indicaciones que debe tener en cuenta son:

- 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 Database para Linux, UNIX y Windows, o para tratar datos GRAPHIC en cualquier aplicación (incluidas las aplicaciones de un solo byte) conectada a una base de datos UCS-2.
- Los caracteres que no sean DBCS, y los caracteres amplios que se pueden convertir en caracteres no DBCS, no se deben utilizar en series gráficas. *Caracteres que no sean DBCS* hace referencia a los caracteres de un solo byte y a los caracteres que no son de doble byte. Las series gráficas no se validan para asegurar que sus valores únicamente contienen puntos de código de caracteres de doble byte. Las variables del lenguaje principal de gráficos sólo deben contener datos DBCS o, si WCHARTYPE CONVERT está en vigor, datos de caracteres amplios que se convierten en datos DBCS. Debe almacenar los datos que contienen una mezcla de caracteres de doble byte y de un solo byte en

variables del lenguaje principal de tipo carácter. Observe que las variables del lenguaje principal de datos mixtos no se ven afectadas por el establecimiento de la opción WCHARTYPE.

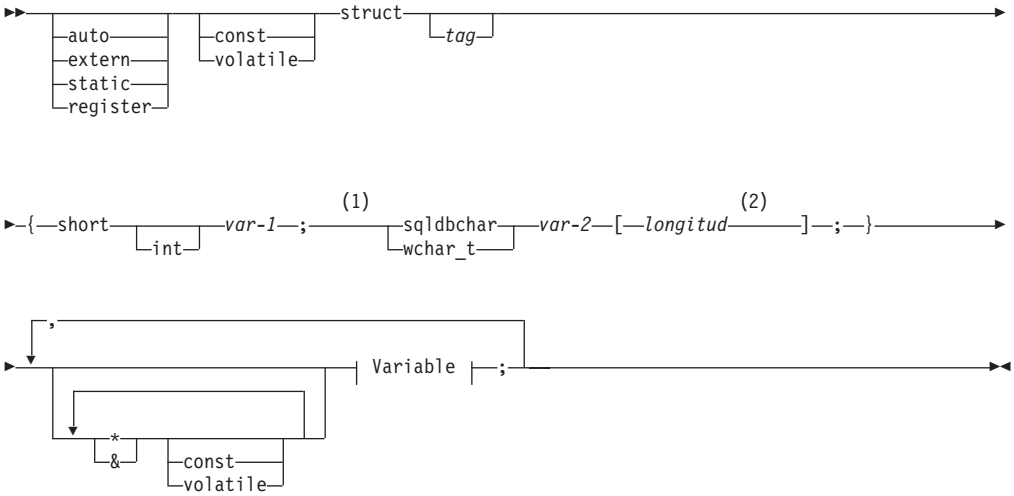
- En las aplicaciones en que se utiliza la opción de precompilación WCHARTYPE NOCONVERT, no se deben utilizar literales L junto con variables del lenguaje principal de gráficos, puesto que los literales L están en formato de caracteres amplios. Un literal L es una serie de caracteres amplios C que tiene como prefijo la letra L y como tipo de datos `"array of wchar_t"`. Por ejemplo, `L"dbcs-string"` es un literal L.
- En las aplicaciones en que se utiliza la opción de precompilación WCHARTYPE CONVERT, se pueden utilizar literales L para inicializar variables del lenguaje principal `wchar_t`, pero no se pueden utilizar en sentencias de SQL. En lugar de utilizar literales L, las sentencias de SQL deben usar constantes de serie de gráficos, que son independientes del valor de WCHARTYPE.
- El valor de la opción WCHARTYPE afecta a los datos de gráficos que se pasan al gestor de bases de datos y que se reciben de éste utilizando la estructura SQLDA, así como a las variables del lenguaje principal. Si WCHARTYPE CONVERT está en vigor, se supondrá que los datos gráficos recibidos de la aplicación mediante una SQLDA están en formato de caracteres amplios, y se convertirán al formato DBCS a través de una llamada implícita a `wcstombs()`. De forma parecida, los datos de salida de gráficos recibidos por una aplicación se habrán convertido al formato de caracteres amplios antes de colocarlos en el almacenamiento de la aplicación.
- Los procedimientos almacenados sin barrera se deben precompilar con la opción WCHARTYPE NOCONVERT. Los procedimientos almacenados normales con barrera se pueden precompilar con las opciones CONVERT o NOCONVERT, lo cual afectará al formato de los datos gráficos manipulados por las sentencias de SQL contenidas en el procedimiento almacenado. No obstante, y en cualquier caso, los datos gráficos que se pasen al procedimiento almacenado mediante la SQLDA estarán en formato DBCS. De la misma forma, los datos que se pasen desde el procedimiento almacenado mediante la SQLDA deben estar en formato DBCS.
- Si una aplicación llama a un procedimiento almacenado a través de la interfaz Database Application Remote Interface (DARI) (la API `sqleproc()`), los datos gráficos de la SQLDA de entrada tienen que estar en formato DBCS, o en UCS-2 si está conectada a una base de datos UCS-2, independientemente del estado del valor de WCHARTYPE de la aplicación que realiza la llamada. Asimismo, los datos gráficos de la SQLDA de salida se devolverán en formato DBCS, o en UCS-2 si está conectada a una base de datos UCS-2, independientemente del valor de WCHARTYPE.
- Si una aplicación llama a un procedimiento almacenado mediante la sentencia CALL de SQL, se producirá una conversión de los datos gráficos en la SQLDA, según el valor de WCHARTYPE de la aplicación que realiza la llamada.
- Los datos gráficos que se pasen a funciones definidas por el usuario (UDF) siempre estarán en formato DBCS. De la misma forma, se supondrá que los datos gráficos devueltos desde una UDF están en formato DBCS para las bases de datos DBCS, y en formato UCS-2 para las bases de datos EUC y UCS-2.
- Los datos almacenados en archivos DBCLOB mediante el uso de variables de referencia a archivos DBCLOB se almacenan en formato DBCS o, en el caso de bases de datos UCS-2, en formato UCS-2. Del mismo modo, los datos de entrada procedentes de archivos DBCLOB se recuperan en formato DBCS o, en el caso de bases de datos UCS-2, en formato UCS-2.

Nota:

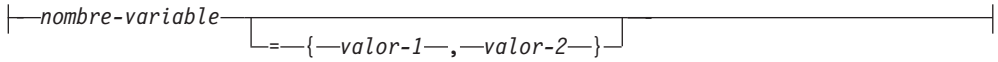
1. En el caso de DB2 para sistemas operativos Windows, la opción `WCHARTYPE CONVERT` está soportada para aplicaciones compiladas con el compilador Microsoft Visual C++. Sin embargo, no utilice la opción `CONVERT` con este compilador si la aplicación inserta datos en una base de datos DB2 en una página de códigos que sea diferente a la de la base de datos. En esta situación, el servidor DB2 suele efectuar una conversión de página de códigos; no obstante, el entorno de tiempo de ejecución de Microsoft C no maneja caracteres de sustitución para algunos caracteres de doble byte. Esto podría producir errores de conversión en tiempo de ejecución.
2. 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.

Declaración de variables del lenguaje principal de tipo VARGRAPHIC en el formato estructurado en aplicaciones de SQL incorporado C o C++

A continuación se muestra la sintaxis para declarar una variable del lenguaje principal gráfica mediante el 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

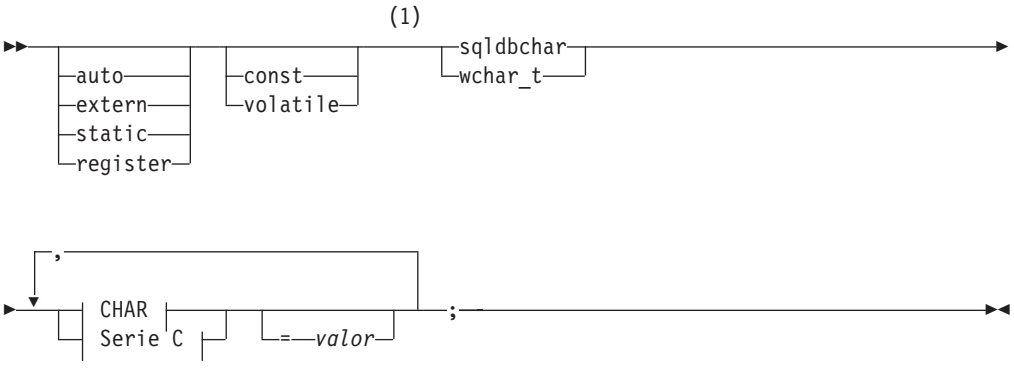
VARGRAPHIC (SQLTYPE 464) o LONG VARGRAPHIC (SQLTYPE 472). El valor de longitud debe ser mayor o igual que 1 y menor o igual que la longitud máxima de LONG VARGRAPHIC, que es 16350.

Consideraciones sobre la declaración de gráfico (formato estructurado VARGRAPHIC):

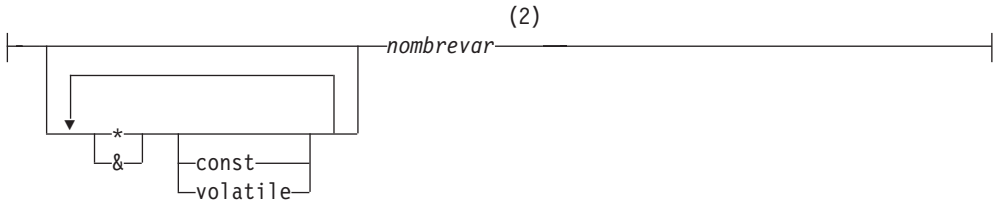
1. *var-1* y *var-2* deben ser simples referencias a variables (y no operadores) y no se pueden utilizar como variables del lenguaje principal.
2. *valor-1* y *valor-2* son inicializadores de *var-1* y *var-2*. *valor-1* debe ser un entero y *valor-2* debe ser un literal de serie de caracteres amplios (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.

Declaración de variables de tipo GRAPHIC del lenguaje principal en formatos de un solo gráfico y de gráficos terminados en nulo en aplicaciones de SQL incorporado C y C++

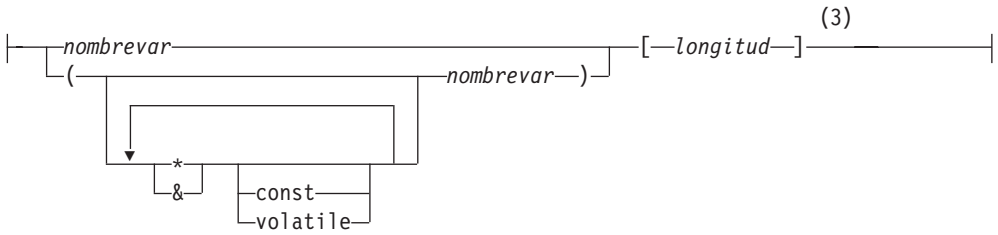
A continuación se muestra la sintaxis para declarar una variable del lenguaje principal gráfica mediante el formato de un solo gráfico y el formato de gráfico terminado en nulo.



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

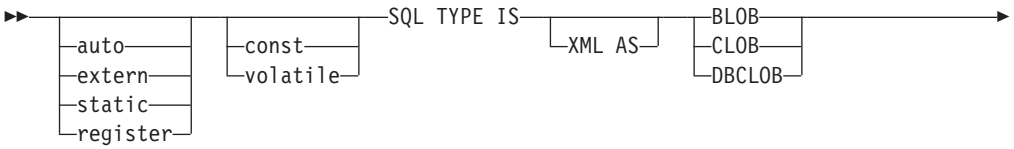
3 Serie gráfica terminada en nulo (SQLTYPE 400)

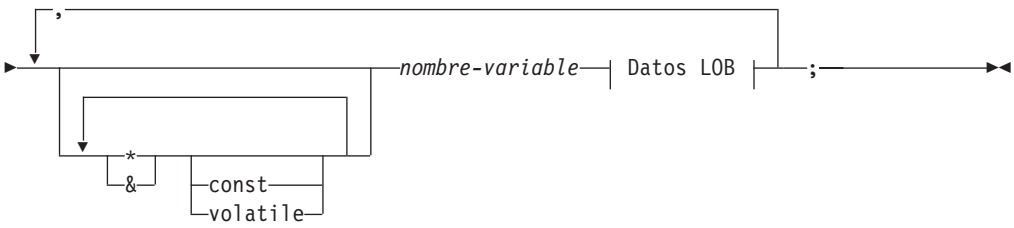
Consideraciones sobre variables del lenguaje principal de tipo graphic:

1. El formato de un solo gráfico 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 16 336.
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.

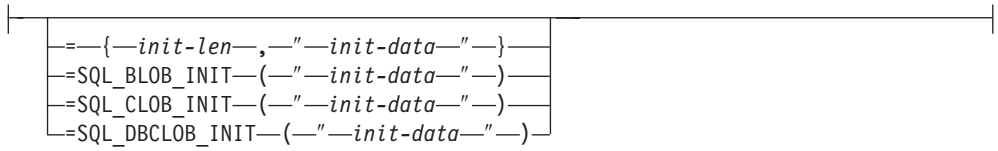
Declaración de variables del lenguaje principal de tipo objeto grande (large object) en aplicaciones de SQL incorporado C y C++

A continuación se muestra la sintaxis para declarar 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 *length* tras la evaluación de DBCLOB debe ser 1 <= longitud <= 1.073.741.823.

Consideraciones sobre variables del lenguaje principal de tipo LOB:

1. Es necesaria la cláusula TYPE IS de SQL para poder distinguir los tres tipos de LOB entre sí, de forma que se puedan llevar a cabo una comprobación del tipo y una resolución de la función para las variables de lenguaje principal de tipo LOB que se pasan a las funciones.
2. SQL TYPE IS, BLOB, CLOB, DBCLOB, K, M, G se pueden especificar en una mezcla de mayúsculas y minúsculas.
3. La longitud máxima permitida para la serie de inicialización *"init-data"* es 32702 bytes, incluidos los delimitadores de la serie (igual al límite existente en series C y C++ dentro del precompilador).
4. La longitud de inicialización, *long-inic*, tiene que ser una constante numérica (por ejemplo, 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:

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:

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:

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;
```

Declaración:

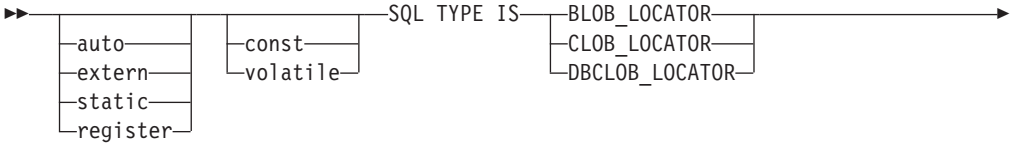
```
SQL TYPE IS DBCLOB(30000) my_dbclob2 = SQL_DBCLOB_INIT(L"mydbdata");
```

Si se precompila con la opción WCHARTYPE CONVERT, tiene como resultado la generación de la estructura siguiente:

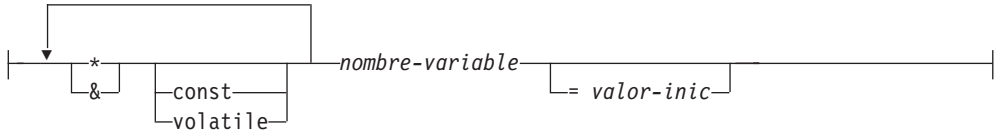
```
struct my_dbclob2_t {
     sqluint32        length;
     wchar_t          data[30000];
} my_dbclob2 = SQL_DBCLOB_INIT(L"mydbdata");
```

Declaración de variables del lenguaje principal de tipo localizador de objeto grande (large object locator) en aplicaciones de SQL incorporado C y C++

A continuación se muestra la sintaxis para declarar variables del lenguaje principal de localizador de objeto grande (LOB) en C o C++.



Variable



Consideraciones sobre variables del lenguaje principal de tipo 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 (las declaraciones de otros tipos de localizadores de LOB son similares):

Declaración:

```
SQL TYPE IS CLOB_LOCATOR my_locator;
```

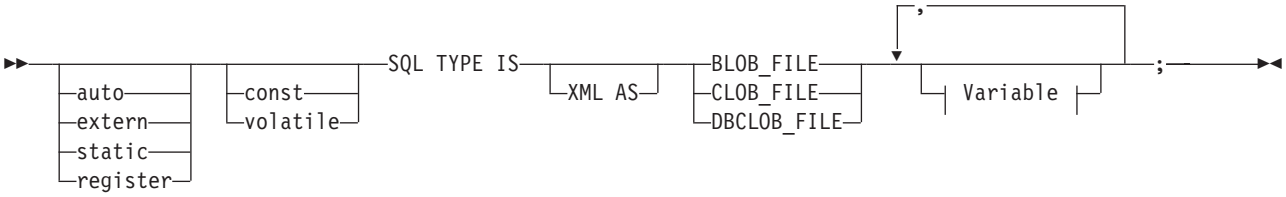
Tiene como resultado la generación de la declaración siguiente:

```
sqluint32 my_locator;
```

Declaración de variables del lenguaje principal de tipo referencia de archivos en aplicaciones de SQL incorporado C y C++

A continuación se muestra la sintaxis para declarar variables del lenguaje principal de referencia de archivos en C o C++.

Sintaxis para variables del lenguaje principal de referencia de archivos en C o C++



Variable

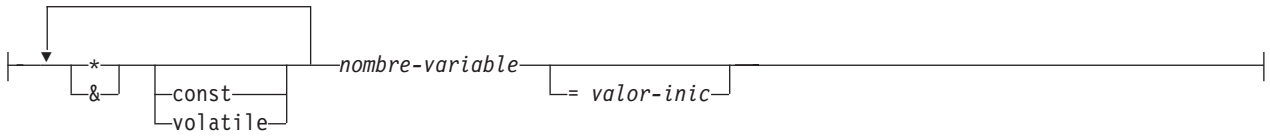


Figura 1. Diagrama de sintaxis

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 (las declaraciones de otros tipos de referencias de archivos LOB son similares):

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;
```

Nota: La estructura anterior es equivalente a la estructura de sqlfile que se encuentra en la cabecera sql.h. Consulte la Figura 1 en la página 88 para ver el diagrama de sintaxis.

Declaración de variables del lenguaje principal como punteros en aplicaciones de SQL incorporado C y C++

Las variables del lenguaje principal se pueden declarar como punteros a tipos de datos específicos, con las restricciones siguientes:

- Si una variable del lenguaje principal se declara como puntero, no se puede declarar ninguna otra variable del lenguaje principal con el mismo nombre dentro del mismo archivo fuente. El ejemplo siguiente no está permitido:

  ```
  char mystring[20];
  char (*mystring)[20];
  ```

- Cuando declare un puntero a una matriz de caracteres terminada en nulo, utilice paréntesis. En todos los casos restantes, no se permiten paréntesis. Por ejemplo:

  ```
  EXEC SQL BEGIN DECLARE SECTION;
    char  (*arr)[10];  /* correcto  */
    char  *(arr);      /* incorrecto */
    char  *arr[10];    /* incorrecto */
  EXEC SQL END DECLARE SECTION;
  ```

  La primera declaración es un puntero a una matriz de caracteres de 10 bytes. Esta variable del lenguaje principal es válida. La segunda declaración no es válida. Los paréntesis no están permitidos en un puntero a un carácter. La tercera declaración es una matriz de punteros. Este tipo de datos no se soporta.

  La declaración de variable del lenguaje principal:

  ```
  char *ptr;
  ```

  se acepta, pero no significa *serie de caracteres terminada en nulo de longitud indeterminada*; significa *puntero a una variable del lenguaje principal de un solo carácter y de longitud fija*. Es posible que esto no fuera lo que se pretendía. Para definir una variable del lenguaje principal de puntero que pueda indicar distintas series de caracteres, utilice el primero de los formatos de declaración anteriores.

- Cuando se utilizan variables del lenguaje principal de puntero en sentencias de SQL, deben tener como prefijo el mismo número de asteriscos con que se han declarado, tal como en el ejemplo siguiente:

  ```
  EXEC SQL BEGIN DECLARE SECTION;
    char (*mychar)[20];    /* Puntero a matriz de caracteres de 20 bytes */
  EXEC SQL END DECLARE SECTION;

  EXEC SQL SELECT column INTO :*mychar FROM table;   /* Correcta */
  ```

- Sólo se puede utilizar el asterisco como operador en un nombre de variable del lenguaje principal.
- La longitud máxima de un nombre de variable del lenguaje principal no se ve afectada por el número de asteriscos especificados, puesto que no se considera que los asteriscos formen parte del nombre.
- Siempre que utilice una variable de puntero en una sentencia de SQL, debe dejar la opción de precompilación del nivel de optimización (OPTLEVEL) en su valor por omisión, que es 0 (sin optimización). Esto significa que el gestor de bases de datos no realizará ninguna optimización de la SQLDA.

Declaración de miembros de datos de clase como variables del lenguaje principal en aplicaciones de SQL incorporado 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).

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;
      }
};
```

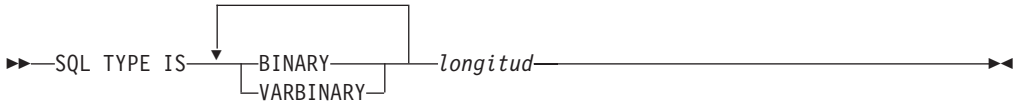
En este ejemplo, los miembros de datos de clase `staff_name`, `staff_id` y `staff_salary` se utilizan directamente en la sentencia INSERT. Puesto que se han declarado como variables del lenguaje principal (consulte el primer ejemplo de esta sección), están calificados de forma implícita para el objeto 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;
}
```

Declaración de variables del lenguaje principal de tipo binario en aplicaciones de SQL incorporado C y C++

A continuación se muestra la sintaxis para declarar variables del lenguaje principal de localizador varbinary en C o C++.



Ejemplo

Declaración:

```
SQL TYPE IS BINARY(4) myBinField;
```

Tiene como resultado la generación del código C siguiente:

```
 unsigned char myBinField[4];

where length  N (1<= N <=255)
```

Declaración:

```
SQL TYPE IS VARBINARY(12) myVarBinField;
```

Tiene como resultado la generación del código C siguiente:

```
struct myVarBinField_t { sqluint16 length;
char data[12];
} myVarBinField;

Where length is N (1<= N <=32704)
```

Soporte de BINARY y VARBINARY en aplicaciones de SQL incorporado

Para utilizar los tipos de BINARY y VARBINARY en aplicaciones de SQL incorporado, utilice el tipo de datos adecuado como se muestra en la sección de declaración. Para los datos BINARY, copie los datos de la variable definida por el usuario y utilice la variable de las sentencias SQL. Para los datos VARBINARY, establezca la longitud en el valor adecuado antes de copiar los datos.

El ejemplo siguiente muestra cómo utilizar los dos tipos de datos en aplicaciones de SQL incorporado:

```
EXEC SQL BEGIN DECLARE SECTION;
sql type is binary(50)  binary1 ;
sql type is varbinary(100)  binary2 ;
```

```
EXEC SQL END DECLARE SECTION;
char  strng1[50];
char  strng2[50];

memset( binary1, 0x00, sizeof(binary1) );
memset( binary2.data, 0x00, sizeof(binary2.data) );
strcpy( strng1, "AAAAAAZZZZZMMMMMMMMMJJJJJJJJJJJJJJ" );
strcpy( strng2, "BBBBBBBBBBBBBBBCCCCCCCCCCCDDDDDDDDEEEEEEEEEEK" );
memcpy( binary1, strng1, strlen(strng1) );
memcpy( binary2.data, strng2, strlen(strng2) );
binary2.length = strlen(binary2.data);
EXEC SQL INSERT INTO test1 VALUES ( :binary1, :binary2 );
```

Cuando se recupera de la base de datos, la longitud de los datos se establece correctamente en la estructura correspondiente.

Resolución de ámbito y operaciones de miembro de clase en aplicaciones de SQL incorporado C y C++

No puede utilizar el operador de resolución de ámbito de C++ '::', ni los operadores de miembro de C y 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';
```

Consideraciones sobre EUC en japonés o chino tradicional y UCS-2 en aplicaciones de SQL incorporado C y C++

Si la página de códigos de la aplicación es EUC en japonés o chino tradicional, o si la aplicación conecta con una base de datos UCS-2, puede acceder a columnas GRAPHIC del servidor de bases de datos utilizando las opciones CONVERT o NOCONVERT y las variables del lenguaje principal de gráficos `wchar_t` o `sqldbchar`, o las SQLDA de entrada/salida. En este apartado, *formato DBCS* hace referencia al esquema de codificación de UCS-2 para datos EUC. Considere los casos siguientes:

- Se utiliza la opción CONVERT

  El cliente DB2 convierte los datos gráficos del formato de caracteres amplios a la página de códigos de la aplicación, y luego a UCS-2, antes de enviar la SQLDA de entrada al servidor de bases de datos. Los datos gráficos se envían al servidor de bases de datos identificados por el identificador de página de códigos de UCS-2. Los datos de tipo carácter mixtos se identifican con el identificador de página de códigos de la aplicación. Cuando un cliente recupera datos gráficos de una base de datos, éstos se identifican con el identificador de página de códigos de UCS-2. El cliente DB2 convierte los datos de UCS-2 a la página de códigos de la aplicación cliente y luego al formato de caracteres amplios. Si se utiliza una SQLDA de entrada en lugar de una variable del lenguaje principal, se requiere al usuario que se asegure de que los datos gráficos se han codificado utilizando el formato de caracteres amplios. Estos datos se convertirán a UCS-2 y luego se enviarán al servidor de bases de datos. Estas conversiones tendrán impacto en el rendimiento.
- Se utiliza la opción NOCONVERT

DB2 supone que los datos se han codificado utilizando UCS-2 y están identificados por la página de códigos de UCS-2, y no se produce ninguna conversión. DB2 supone que la variable del lenguaje principal de gráficos se utiliza simplemente como contenedor. Si no se selecciona la opción NOCONVERT, los datos gráficos recuperados del servidor de bases de datos se pasan a la aplicación codificados mediante UCS-2. Las posibles conversiones de la página de códigos de la aplicación a UCS-2 y de UCS-2 a la página de códigos de la aplicación son responsabilidad del usuario. Los datos identificados como UCS-2 se envían al servidor de bases de datos sin que se produzca ninguna conversión ni alteración.

Para minimizar las conversiones, puede utilizar la opción NOCONVERT y manejar las conversiones en la aplicación, o bien no utilizar columnas GRAPHIC. Para los entornos de cliente en que la codificación `wchar_t` es en Unicode de dos bytes, por ejemplo en Windows 2000 o AIX versión 5.1 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, los sistemas de bases de datos DB2 utilizan `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 gráficas de lenguaje principal después de una conversión a UCS-2 (si se especifica NOCONVERT) ni mediante una conversión al formato de caracteres amplios (si se especifica CONVERT). Esto es así porque los caracteres de estos juegos de códigos EUC pasan a tener un solo byte cuando se convierten de UCS-2 a DBCS de PC.

En general, aunque eucJP y eucTW almacenan los datos gráficos (GRAPHIC) como UCS-2, los datos GRAPHIC contenidos en estas bases de datos siguen siendo datos eucJP o eucTW no ASCII. Concretamente, cualquier espacio rellenado en estos datos GRAPHIC es espacio DBCS (también conocido como espacio ideográfico en UCS-2, U+3000). Sin embargo, para una base de datos UCS-2, los datos GRAPHIC pueden contener cualquier carácter UCS-2 y el relleno del espacio se realiza con espacio UCS-2, U+0020. Tenga presente esta diferencia cuando codifique aplicaciones para recuperar datos UCS-2 de una base de datos UCS-2, en comparación con la recuperación de datos UCS-2 de bases de datos eucJP y eucTW.

Almacenamiento binario de valores de variables mediante la cláusula FOR BIT DATA en aplicaciones de SQL incorporado 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).

Inicialización de variables del lenguaje principal en aplicaciones de SQL incorporado C y C++

En las secciones declare de C y C++, puede declarar e inicializar varias variables en una sola línea. Sin embargo, las variables se deben inicializar utilizando el símbolo "=", no 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 y DECLARE SECTION de aplicaciones de SQL incorporado C y C++

El precompilador C o C++ no puede procesar directamente ninguna macro de C utilizada en una declaración dentro de una sección de declaración. En lugar de esto, en primer lugar se debe preprocesar el archivo fuente mediante un preprocesador de C externo. Para ello, especifique el mandato exacto para invocar un preprocesador de C para el precompilador mediante la opción PREPROCESSOR.

Cuando se especifica la opción PREPROCESSOR, el precompilador procesa en primer lugar todas las sentencias de SQL INCLUDE, incorporando al archivo fuente el contenido de todos los archivos a los que se hace referencia en la sentencia de SQL INCLUDE. A continuación, el precompilador invoca al preprocesador de C externo, utilizando el mandato que se especifique con el archivo fuente modificado como entrada. El archivo preprocesado, del cual el precompilador siempre espera que tenga la extensión `.i`, se utiliza como el nuevo archivo fuente para el resto del proceso de precompilación.

Cualquier macro #line generada por el precompilador deja de hacer referencia al archivo fuente original, pasando a hacerla al archivo preprocesado. Para relacionar los errores del compilador con el archivo fuente original, mantenga comentarios en el archivo preprocesado. Le serán de ayuda para localizar diversas secciones de los archivos fuente originales, incluidos los archivos de cabecera. La opción para mantener comentarios suele estar disponible en los preprocesadores C y la puede incluir en el mandato que especifique mediante la opción PREPROCESSOR. No debe hacer que el preprocesador de C produzca por sí mismo como salida ninguna macro #line, puesto que se podrían mezclar incorrectamente con las generadas por el precompilador.

Notas sobre la utilización de la expansión de macros:

1. El mandato que especifique mediante la opción PREPROCESSOR debe incluir todas las opciones deseadas, pero no el nombre del archivo de entrada. Por ejemplo, para IBM C 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 la sección de declaración de aplicaciones de SQL incorporado C y C++

Con el soporte de estructuras del lenguaje principal, el precompilador C o C++ permite agrupar las variables del lenguaje principal en una sola estructura del lenguaje principal. Esta función brinda una nota taquigráfica para hacer referencia a ese mismo conjunto de variables del lenguaje principal en una sentencia de SQL. Por ejemplo, se puede utilizar la siguiente estructura del lenguaje principal para acceder a algunas de las columnas de la tabla STAFF de la base de datos SAMPLE:

```
struct tag
   {
     short id;
     struct
     {
       short length;
       char  data[10];
     } name;
     struct
     {
       short   years;
       double    salary;
     } info;
   } staff_record;
```

Los campos de una estructura del lenguaje principal pueden ser de cualquiera de los tipos de variable del lenguaje principal válidos. Los tipos válidos incluyen todos los tipos numéricos, de carácter y de objetos grande. También se soportan estructuras del lenguaje principal anidadas hasta 25 niveles. En el ejemplo anterior, el campo `info` es una subestructura, mientras que el campo `name` no lo es, puesto que representa un campo VARCHAR. Se aplica el mismo principio a LONG VARCHAR, VARGRAPHIC y LONG VARGRAPHIC. Asimismo, se soportan punteros a las estructuras del lenguaje principal.

Existen dos maneras de hacer referencia a las variables del lenguaje principal agrupadas en una estructura del lenguaje principal en una sentencia de SQL:

- Se puede hacer referencia al nombre de la estructura del lenguaje principal en una sentencia de SQL.

  ```
  EXEC SQL SELECT id, name, years, salary
       INTO :staff_record
       FROM staff
       WHERE id = 10;
  ```

El precompilador convierte la referencia a `staff_record` en una lista de todos los campos declarados en la estructura del lenguaje principal, separados por comas. Cada campo se califica con los nombres de estructura del lenguaje principal de todos los niveles, a fin de evitar conflictos de denominación con otras variables del lenguaje principal o campos. Esto es equivalente al método siguiente.

- Se puede hacer referencia a nombres de estructuras del lenguaje principal completamente calificadas en una sentencia de SQL.

```
EXEC SQL SELECT id, name, years, salary
     INTO :staff_record.id, :staff_record.name,
          :staff_record.info.years, :staff_record.info.salary
     FROM staff
     WHERE id = 10;
```

  Las referencias a nombres de campos se deben calificar por completo aunque no existan otras variables del lenguaje principal que tengan el mismo nombre. También se puede hacer referencia a subestructuras calificadas. En el ejemplo anterior, se puede utilizar `:staff_record.info` para sustituir a `:staff_record.info.years, :staff_record.info.salary`.

Puesto que una referencia a una estructura del lenguaje principal (primer ejemplo) es equivalente a una lista de sus campos, separados por comas, existen casos en que este tipo de referencia puede conducir a un error. Por ejemplo:

```
EXEC SQL DELETE FROM :staff_record;
```

Aquí, la sentencia DELETE espera una sola variable del lenguaje principal basada en caracteres. Si en su lugar se proporciona una estructura del lenguaje principal, la sentencia tiene como resultado un error durante la precompilación:

```
SQL0087N  La variable del lenguaje principal "staff_record" es una estructura
que se utiliza donde no se permiten referencias a estructuras.
```

Otros usos de las estructuras del lenguaje principal que pueden ocasionar que se produzca un error SQL0087N incluyen: PREPARE, EXECUTE IMMEDIATE, CALL, variables de indicador y referencias a SQLDA. Las estructuras del lenguaje principal que tengan exactamente un campo están permitidas en estas situaciones, puesto que constituyen referencias a campos individuales (segundo ejemplo).

Variables de indicador de nulo o de truncamiento y tablas de indicadores en aplicaciones de SQL incorporado C y C++

Para cada variable del lenguaje principal que pueda recibir valores nulos, declare **variables de indicador** como un tipo de datos `short`.

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]`.

Series terminadas en nulo en aplicaciones de SQL incorporado C y C++

Las series terminadas en nulo de C y C++ tienen su propio SQLTYPE (460/461 para caracteres y 468/469 para gráficos).

Las series terminadas en nulo de C y C++ se manejan de forma distinta en función del valor de la opción LANGLEVEL del precompilador. Si se especifica una variable del lenguaje principal con uno de estos valores SQLTYPE y la longitud declarada *n* dentro de una sentencia de SQL y el número de bytes de datos (para tipos de carácter) o de caracteres de doble byte (para tipos gráficos) es *k*, sucede lo siguiente:

- Si la opción LANGLEVEL del mandato PREP es SAA1 (valor por omisión):

Para la salida:

Si...	Sucede que...
k > *n*	Se mueven *n* caracteres a la variable del lenguaje principal de destino, SQLWARN1 se establece en 'W' y SQLCODE es 0 (SQLSTATE 01004). No se coloca ningún terminador nulo en la serie. Si se ha especificado una variable de indicador con la variable del lenguaje principal, el valor de la variable de indicador se establece en *k*.
k = *n*	Se mueven *k* caracteres a la variable del lenguaje principal de destino, SQLWARN1 se establece en 'N' y SQLCODE es 0 (SQLSTATE 01004). No se coloca ningún terminador nulo en la serie. Si se ha especificado una variable de indicador con la variable del lenguaje principal, el valor de la variable de indicador se establece en 0.
k < *n*	Los caracteres *k* se mueven a la variable del lenguaje principal de destino y, en lugar del carácter *k + 1* se inserta un carácter 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 0.

Para entrada:

Cuando el gestor de bases de datos encuentre una variable del lenguaje principal de entrada que tiene uno de estos valores SQLTYPE y no finaliza por un terminador nulo, supondrá que el carácter *n+1* contendrá el carácter terminador nulo.

- Si la opción LANGLEVEL del mandato PREP es MIA:

  Para 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 01501). 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 entrada:

  Cuando el gestor de bases de datos encuentre una variable del lenguaje

principal de entrada que tenga uno de estos valores SQLTYPE y no finalice por un carácter nulo, se devolverá SQLCODE -302 (SQLSTATE 22501).

Si se especifica en cualquier otro contexto de SQL, una variable del lenguaje principal con SQLTYPE 460 y longitud *n* se trata como el tipo de datos VARCHAR con longitud *n*, tal como se ha definido anteriormente. Si se especifica en cualquier otro contexto de SQL, una variable del lenguaje principal con SQLTYPE 468 y longitud *n* se trata como el tipo de datos VARGRAPHIC con longitud *n*, tal como se ha definido anteriormente.

Variables del lenguaje principal en COBOL

Las variables del lenguaje principal son variables del lenguaje COBOL a las que se hace referencia en las sentencias de SQL. Permiten a una aplicación intercambiar datos con el gestor de bases de datos. Una vez que se ha precompilado la aplicación, el compilador utiliza las variables del lenguaje principal como cualquier otra variable de COBOL. Cuando denomine, declare y utilice variables del lenguaje principal, siga las normas descritas en los apartados siguientes.

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 siguientes normas:

- 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.

Sección declare para variables del lenguaje principal en aplicaciones de SQL incorporado 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. Por ejemplo:

```
    EXEC SQL BEGIN DECLARE SECTION END-EXEC.
77 dept              pic s9(4) comp-5.
01 userid            pic x(8).
01 passwd.
    EXEC SQL END DECLARE SECTION END-EXEC.
```

El precompilador COBOL sólo reconoce un subconjunto de las declaraciones válidas de COBOL.

Ejemplo: Plantilla de sección de declaración de SQL para aplicaciones de SQL incorporado 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.          /* SQL tipo  500 */
  01 divis      PIC S9(9) COMP-5.          /* SQL tipo  496 */
  01 salary     PIC S9(6)V9(3) COMP-3.     /* SQL type  484 */
  01 bonus      USAGE IS COMP-1.           /* SQL tipo  480 */
  01 wage       USAGE IS COMP-2.           /* SQL tipo  480 */
  01 nm         PIC X(5).                  /* SQL tipo  452 */
  01 varchar.
     49 leng    PIC S9(4) COMP-5.          /* SQL tipo  448 */
     49 strg    PIC X(14).                 /* SQL tipo  448 */
  01 longvchar.
     49 len     PIC S9(4) COMP-5.          /* SQL tipo  456 */
     49 str     PIC X(6027).               /* SQL tipo  456 */
  01 MY-CLOB USAGE IS SQL TYPE IS CLOB(1M).                 /* SQL tipo  408 */
  01 MY-CLOB-LOCATOR USAGE IS SQL TYPE IS CLOB-LOCATOR.     /* SQL tipo  964 */
  01 MY-CLOB-FILE USAGE IS SQL TYPE IS CLOB-FILE.           /* SQL tipo  920 */
  01 MY-BLOB USAGE IS SQL TYPE IS BLOB(1M).                 /* SQL tipo  404 */
  01 MY-BLOB-LOCATOR USAGE IS SQL TYPE IS BLOB-LOCATOR.     /* SQL tipo  960 */
  01 MY-BLOB-FILE USAGE IS SQL TYPE IS BLOB-FILE.           /* SQL tipo  916 */
  01 MY-DBCLOB USAGE IS SQL TYPE IS DBCLOB(1M).             /* SQL tipo  412 */
  01 MY-DBCLOB-LOCATOR USAGE IS SQL TYPE IS DBCLOB-LOCATOR. /* SQL tipo  968 */
  01 MY-DBCLOB-FILE USAGE IS SQL TYPE IS DBCLOB-FILE.       /* SQL tipo  924 */
  01 MY-PICTURE PIC G(16000) USAGE IS DISPLAY-1.            /* SQL tipo  464 */
  01 dt         PIC X(10).                 /* SQL tipo  384 */
  01 tm         PIC X(8).                  /* SQL tipo  388 */
  01 tmstmp     PIC X(26).                 /* SQL tipo  392 */
  01 wage-ind   PIC S9(4) COMP-5.          /* SQL tipo  464 */
*
EXEC SQL END DECLARE SECTION END-EXEC.
```

Tipos de datos BINARY/COMP-4 en aplicaciones de SQL incorporado 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 los ejemplos proporcionados, 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

Variables SQLSTATE y SQLCODE en aplicaciones de SQL incorporado 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:

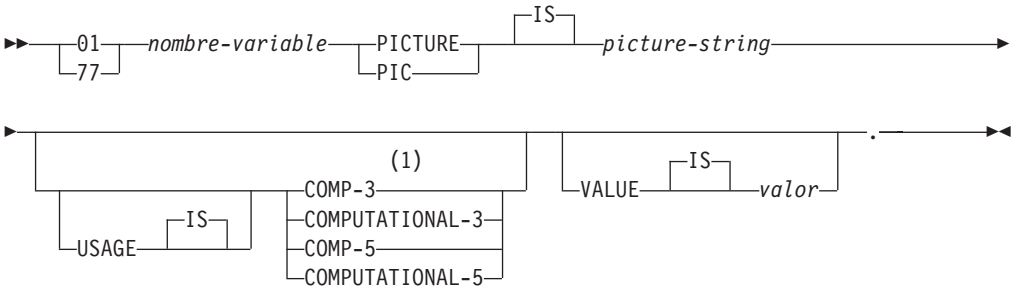
```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 SQLSTATE PIC X(5).
01 SQLCODE  PIC S9(9) USAGE COMP.
.
.
.
EXEC SQL END DECLARE SECTION END-EXEC.
```

Si no se especifica ninguna de ellas, se supone la declaración SQLCODE durante el paso de precompilación. Las variables SQLCODE y SQLSTATE se pueden declarar utilizando el nivel 01 (como se ha mostrado anteriormente) o el nivel 77. 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.

Declaración de variables numéricas del lenguaje principal en aplicaciones de SQL incorporado COBOL

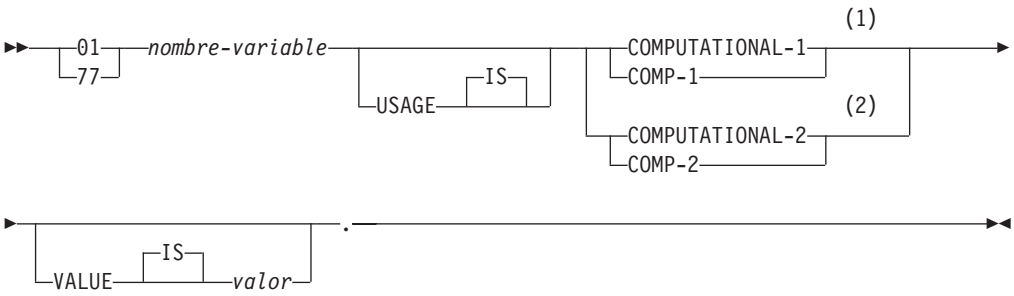
A continuación se muestra la sintaxis de las variables numéricas del lenguaje principal.



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:

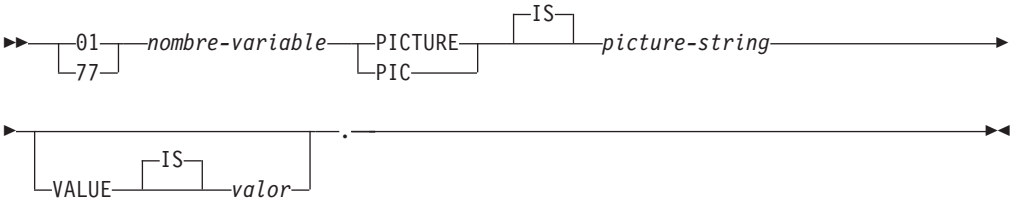
1. *Serie-imagen* debe tener uno de los formatos siguientes:
   - S9(m)V9(n)
   - S9(m)V
   - S9(m)
2. Los nueves se pueden expandir (por ejemplo, ″S999″ en lugar de S9(3)″)

3. *m* y *n* deben ser enteros positivos.

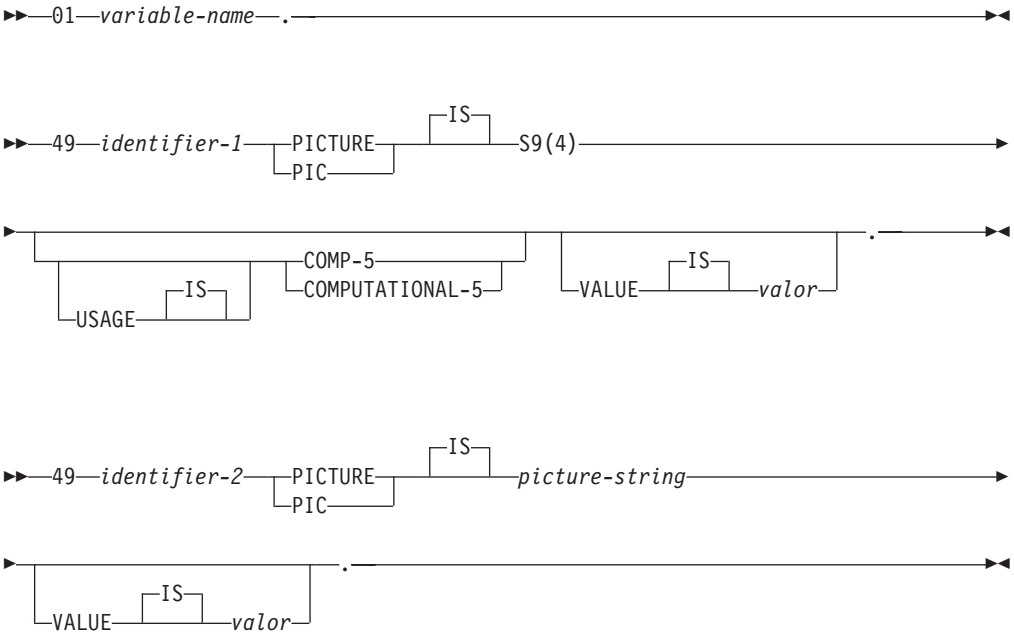
Declaración de variables del lenguaje principal de tipo carácter de longitud fija y longitud variable en aplicaciones de SQL incorporado COBOL

A continuación se muestra la sintaxis de las variables de tipo carácter del lenguaje principal.

Longitud fija



Longitud variable



Consideración 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 fija.
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.
```

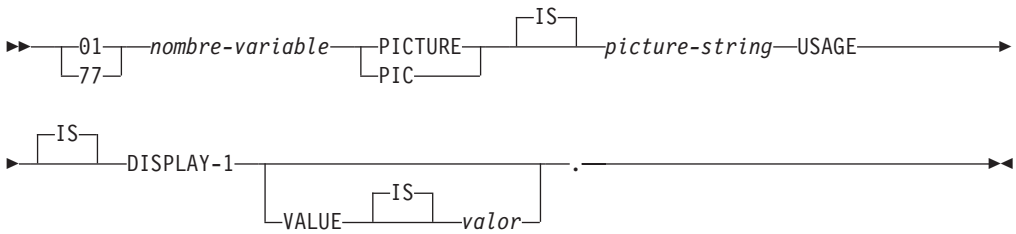
   Sin embargo, y puesto que los espacios en blanco pueden ser significativos en las contraseñas, la variable del lenguaje principal `p-word` se debe declarar como un elemento de datos VARCHAR, de forma que la aplicación pueda indicar explícitamente la longitud significativa de la contraseña para la sentencia 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.
```

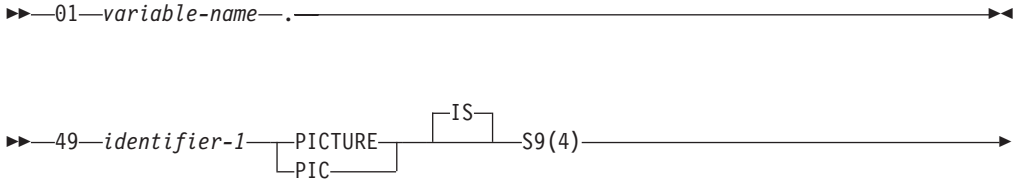
Declaración de variables del lenguaje principal de tipo graphic de longitud fija y longitud variable en aplicaciones de SQL incorporado COBOL

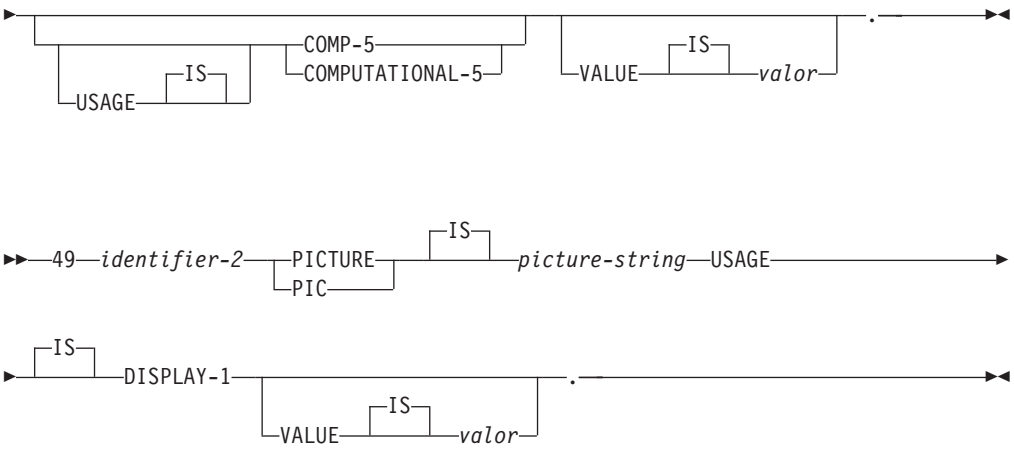
A continuación se muestra la sintaxis de las variables de tipo graphic del lenguaje principal.

Longitud fija



Longitud variable



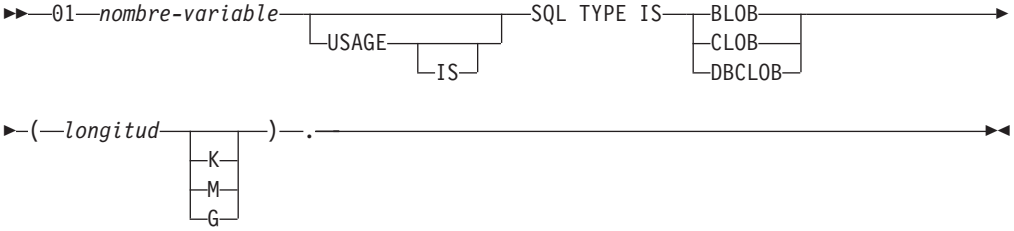


Consideraciones sobre las variables gráficas del lenguaje principal:

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.

Declaración de variables del lenguaje principal de tipo objeto grande (large object) en aplicaciones de SQL incorporado COBOL

A continuación se muestra la sintaxis para declarar variables del lenguaje principal de objeto grande (LOB) en COBOL.



Consideraciones sobre variables del lenguaje principal de tipo 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:

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:

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:

Declaración:

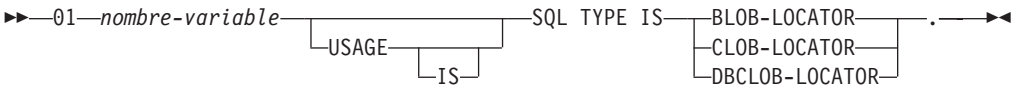
```
01 MY-DBCLOB USAGE IS SQL TYPE IS DBCLOB(30000).
```

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.
```

Declaración de variables del lenguaje principal de tipo localizador de objeto grande (large object locator) en aplicaciones de SQL incorporado COBOL

A continuación se muestra la sintaxis para declarar variables del lenguaje principal de localizador de objeto grande (LOB) en COBOL.



Consideraciones sobre variables del lenguaje principal de tipo 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):

Declaración:

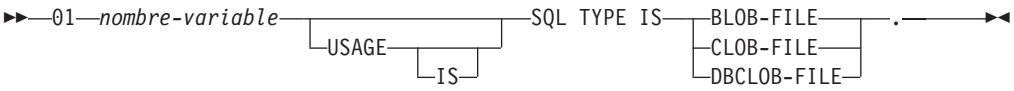
```
01 MY-LOCATOR USAGE SQL TYPE IS BLOB-LOCATOR.
```

Tiene como resultado la generación de la declaración siguiente:

```
01 MY-LOCATOR PIC S9(9) COMP-5.
```

Declaración de variables del lenguaje principal de referencia de archivos aplicaciones de SQL incorporado COBOL

A continuación se muestra la sintaxis para declarar 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):

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).
```

Agrupación de elementos de datos utilizando REDEFINES en aplicaciones de SQL incorporado 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 ...
```

Consideraciones sobre EUC en japonés o chino tradicional y UCS-2 para aplicaciones de SQL incorporado COBOL

Los datos gráficos enviados desde una aplicación que se ejecuta bajo un juego de códigos eucJp o eucTW, o se conecta a una base de datos UCS-2, se identifican con

el identificador de la página de códigos UCS-2. La aplicación debe convertir una serie de caracteres gráficos a UCS-2 antes de enviarla al servidor de bases de datos. Del mismo modo, los datos gráficos recuperados de una base de datos UCS-2 por una aplicación, o de cualquier base de datos por una aplicación que se ejecute bajo una página de códigos EUC eucJP o eucTW, se codifican utilizando UCS-2. Esto requiere que la aplicación realice internamente una conversión de UCS-2 a la página de códigos de la aplicación, a menos que se vayan a presentar datos UCS-2 al usuario.

La aplicación es responsable de convertir a UCS-2 y desde UCS-2, puesto que esta conversión se debe llevar a cabo antes de copiar los datos a la SQLDA o desde esta. DB2 Database para Linux, UNIX y Windows 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.

Almacenamiento binario de valores de variables mediante la cláusula FOR BIT DATA en aplicaciones de SQL incorporado 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.

Soporte de estructuras del lenguaje principal en la sección de declaración de aplicaciones de SQL incorporado 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 método siguiente.

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 variables de indicador de nulo y de indicador de nulo o de truncamiento en aplicaciones de SQL incorporado COBOL

Las variables de indicador de nulo se deben declarar con tipo de datos `PIC S9(4) COMP-5`.

El precompilador COBOL da soporte a la declaración de *tablas de variables de indicador de nulo* (denominadas tablas de indicadores), 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.*

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 a una aplicación intercambiar datos con el gestor de bases de datos. Una vez que se ha precompilado la aplicación, el compilador utiliza las variables del lenguaje principal como cualquier otra variable de FORTRAN. Cuando denomine, declare y utilice variables del lenguaje principal, siga las normas descritas en los apartados siguientes.

Nombres de variables del lenguaje principal en aplicaciones de SQL incorporado FORTRAN

El precompilador SQL identifica las variables del lenguaje principal por el nombre declarado para éstas. Se aplican las siguientes sugerencias:

- 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.

Sección declare para variables del lenguaje principal en aplicaciones de SQL incorporado 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.

Ejemplo: Plantilla de sección de declaración de SQL para aplicaciones de SQL incorporado 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/                    /* SQL type  500 */
  INTEGER*4     DEPT                         /* SQL tipo  496 */
  REAL*4        BONUS                        /* SQL tipo  480 */
  REAL*8        SALARY                       /* SQL tipo  480 */
  CHARACTER     MI                           /* SQL tipo  452 */
  CHARACTER*112 ADDRESS                      /* SQL tipo  452 */
  SQL TYPE IS VARCHAR (512) DESCRIPTION      /* SQL tipo  448 */
  SQL TYPE IS VARCHAR (32000) COMMENTS       /* SQL tipo  448 */
  SQL TYPE IS CLOB (1M) CHAPTER              /* SQL tipo  408 */
  SQL TYPE IS CLOB_LOCATOR CHAPLOC           /* SQL tipo  964 */
```

```
  SQL TYPE IS CLOB_FILE  CHAPFL              /* SQL tipo  920 */
  SQL TYPE IS BLOB (1M) VIDEO                /* SQL tipo  404 */
  SQL TYPE IS BLOB_LOCATOR VIDLOC            /* SQL tipo  960 */
  SQL TYPE IS BLOB_FILE VIDFL                /* SQL tipo  916 */
  CHARACTER*10  DATE                         /* SQL tipo  384 */
  CHARACTER*8   TIME                         /* SQL tipo  388 */
  CHARACTER*26  TIMESTAMP                    /* SQL tipo  392 */
  INTEGER*2     WAGE_IND                     /* SQL tipo  500 */
EXEC SQL END DECLARE SECTION
```

Variables SQLSTATE y SQLCODE en aplicaciones de SQL incorporado 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:

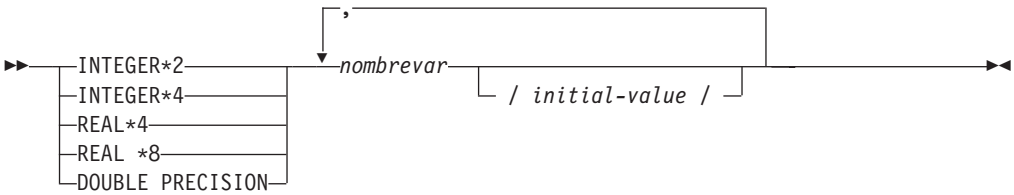
```
EXEC SQL BEGIN DECLARE SECTION;
  CHARACTER*5 SQLSTATE
  INTEGER     SQLCOD
  .
  .
  .
EXEC SQL END DECLARE SECTION
```

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.

Declaración de variables numéricas del lenguaje principal en aplicaciones de SQL incorporado FORTRAN

A continuación se muestra la sintaxis de las variables numéricas del lenguaje principal en FORTRAN.



Consideraciones sobre las variables numéricas del lenguaje principal:

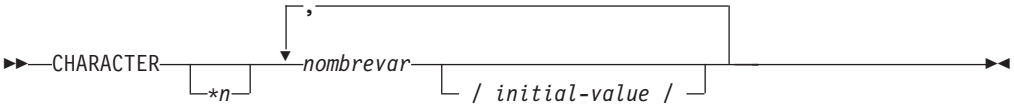
1. REAL*8 y DOUBLE PRECISION son equivalentes.
2. Utilice una E en lugar de una D como indicador de exponente para las constantes REAL*8.

Declaración de variables del lenguaje principal de tipo carácter de longitud fija y longitud variable en aplicaciones de SQL incorporado FORTRAN

A continuación se muestra la sintaxis de las variables de tipo carácter de longitud fija del lenguaje principal.

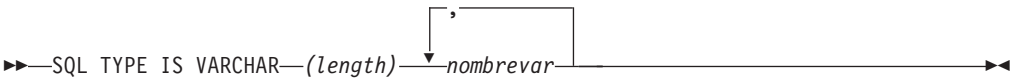
Longitud fija

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 variables del lenguaje principal de tipo carácter:

1. *n tiene un valor máximo de 254.
2. Si la longitud está entre 1 y 32672, ambos inclusive, la variable del lenguaje principal es de tipo VARCHAR(SQLTYPE 448).
3. Si la longitud está entre 32673 y 32700, 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:

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:

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.

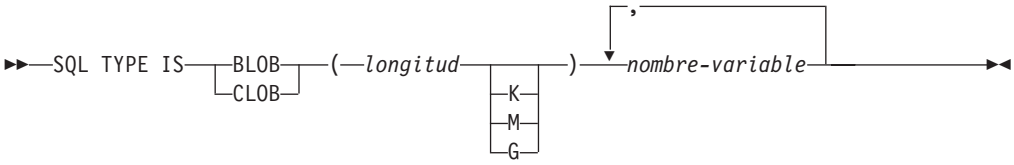
```
EXEC SQL CONNECT TO :dbname USER :userid USING :passwd
```

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
```

Declaración de variables del lenguaje principal de tipo objeto grande (large object) en aplicaciones de SQL incorporado FORTRAN

A continuación se muestra la sintaxis para declarar variables del lenguaje principal de objeto grande (LOB) en FORTRAN.



Consideraciones sobre variables del lenguaje principal de tipo 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:

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:

Declaración:

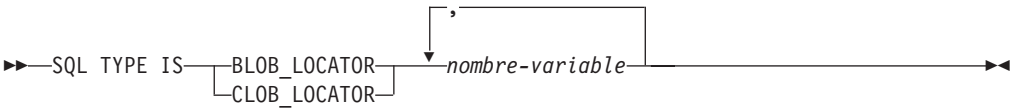
```
sql type is clob(125m) my_clob
```

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 )
```

Declaración de variables del lenguaje principal de tipo localizador de objeto grande (large object locator) en aplicaciones de SQL incorporado FORTRAN

A continuación se muestra la sintaxis para declarar variables del lenguaje principal de localizador de objeto grande (LOB) en FORTRAN.



Consideraciones sobre variables del lenguaje principal de tipo 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):

Declaración:

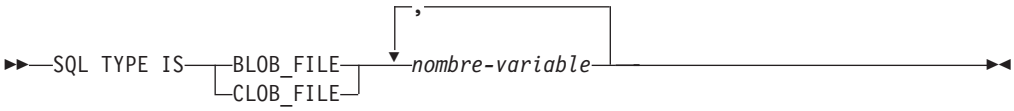
```
SQL TYPE IS CLOB_LOCATOR my_locator
```

Tiene como resultado la generación de la declaración siguiente:

```
integer*4 my_locator
```

Declaración de variables del lenguaje principal de referencia de archivos aplicaciones de SQL incorporado FORTRAN

A continuación se muestra la sintaxis para declarar variables del lenguaje principal de referencia de archivos en FORTRAN.



Consideraciones sobre 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 )
```

Consideraciones sobre los juegos de caracteres gráficos (de varios bytes) en aplicaciones de SQL incorporado 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`. Sin embargo, se puede crear una SQLDA de usuario que contenga datos gráficos.

Consideraciones sobre EUC en japonés o chino tradicional y UCS-2 para aplicaciones de SQL incorporado FORTRAN

Los datos gráficos enviados desde una aplicación que se ejecuta bajo un juego de códigos eucJp o eucTW, o se conecta a una base de datos UCS-2, se identifican con el identificador de la página de códigos UCS-2. La aplicación debe convertir una serie de caracteres gráficos a UCS-2 antes de enviarla al servidor de bases de datos. Del mismo modo, los datos gráficos recuperados de una base de datos UCS-2 por una aplicación, o de cualquier base de datos por una aplicación que se ejecute bajo una página de códigos EUC eucJP o eucTW, se codifican utilizando UCS-2. Esto requiere que la aplicación realice internamente una conversión de UCS-2 a la página de códigos de la aplicación, a menos que se vayan a presentar datos UCS-2 al usuario.

La aplicación es responsable de convertir a UCS-2 y desde UCS-2, puesto que esta conversión se debe llevar a cabo antes de copiar los datos a la SQLDA o desde esta. Los sistemas de bases de datos DB2 no suministran 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.

Variables de indicador de nulo o de truncamiento en aplicaciones de SQL incorporado FORTRAN

Las variables de indicador se deben declarar con tipo de datos `INTEGER*2`.

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 a una aplicación intercambiar datos con el gestor de bases de datos. Una vez que se ha precompilado la aplicación, el compilador utiliza las variables del lenguaje principal como cualquier otra variable de REXX. Cuando denomine, declare y utilice variables del lenguaje principal, siga las normas descritas en los apartados siguientes.

Nombres de variables del lenguaje principal en aplicaciones de SQL incorporado 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 números, caracteres alfabéticos y los caracteres @, _, !, ., ? y $.

Referencias a variables del lenguaje principal en aplicaciones de SQL incorporado 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 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 System i.)

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).

Consideraciones sobre la programación de aplicaciones REXX de SQL incorporado

REXX es un lenguaje interpretado. Por lo tanto, 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.

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.

Utilice los ejemplos siguientes para ver la sintaxis correcta para registrar cada rutina:

Registro de ejemplo en sistemas operativos 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
```

Registro de ejemplo 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.

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.

Declaración de variables del lenguaje principal de tipo objeto grande (large object) en aplicaciones de SQL incorporado REXX

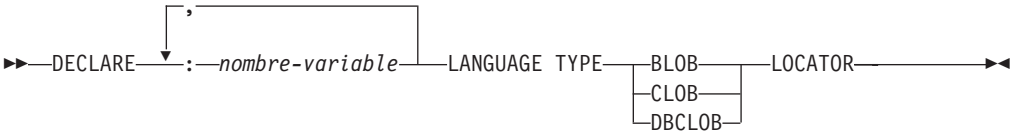
Cuando se capte una columna LOB en una variable del lenguaje principal REXX, esta se almacenará como una serie simple (es decir, descontada). Esto se maneja del mismo modo que todos los tipos de SQL basados en caracteres (tales como CHAR, VARCHAR, GRAPHIC, LONG, etc.). En la entrada, si el tamaño del 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

Declaración de variables del lenguaje principal de tipo localizador de objeto grande (large object locator) en aplicaciones de SQL incorporado REXX

A continuación se muestra la sintaxis para declarar variables del lenguaje principal de localizador de LOB en REXX:



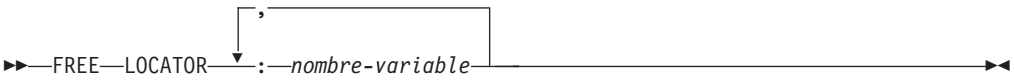
Debe declarar las variables del lenguaje principal de localizador de LOB en la aplicación. Cuando REXX/SQL encuentra estas declaraciones, trata las variables del lenguaje principal declaradas como localizadores durante el resto del programa. Los valores de localizadores se almacenan en variables de REXX, 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



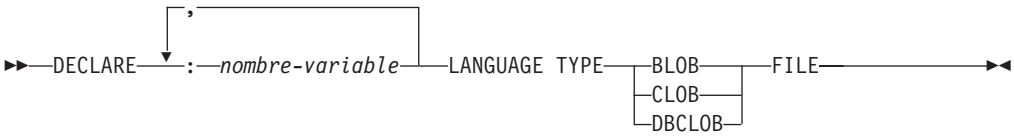
Ejemplo:

```
CALL SQLEXEC 'FREE LOCATOR :hv1, :hv2'
```

Declaración de variables del lenguaje principal de referencia de archivos en aplicaciones de SQL incorporado 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:



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 aplicaciones de SQL incorporado

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).

Variables de indicador de nulo o de truncamiento en aplicaciones de SQL incorporado 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.

Ejecución de expresiones XQuery en aplicaciones de SQL incorporado

Puede almacenar datos XML en las tablas y utilizar aplicaciones de SQL incorporado para acceder a columnas XML mediante expresiones XQuery. Para acceder a datos XML, utilice variables del lenguaje principal XML en lugar de convertir los datos a tipos de datos de carácter o binarios. Si no utiliza las variables del lenguaje principal XML, la mejor alternativa para acceder a datos XML es mediante FOR BIT DATA o con tipos de datos BLOB para evitar conversiones de páginas de códigos.

- Declare las variables del lenguaje principal XML dentro de las aplicaciones de SQL incorporado.
- Se debe utilizar un tipo XML para recuperar valores de XML en una sentencia de SQL estático SELECT INTO.
- Si se utiliza una variable del lenguaje principal CHAR, VARCHAR, CLOB o BLOB para entrada donde se espera un valor XML, el valor estará sujeto a una operación de función XMLPARSE con manejo por omisión de espacios en blanco (STRIP). De lo contrario, se necesita una variable del lenguaje principal XML.

Para ejecutar expresiones XQuery en la aplicación de SQL incorporado directamente, añada al principio de la expresión la palabra clave ″XQUERY″. Para SQL estático, utilice la función XMLQUERY. Cuando se llama a la función XMLQUERY, la expresión XQuery no tiene el prefijo ″XQUERY″.

Ejemplo 1: Ejecución de expresiones XQuery directamente en SQL dinámico C y C++ añadiendo como prefijo la palabra clave ″XQUERY″

En aplicaciones C y C++, las expresiones XQuery se pueden ejecutar del siguiente modo:

```
EXEC SQL INCLUDE SQLCA;
EXEC SQL BEGIN DECLARE SECTION;
  char stmt[16384];
  SQL TYPE IS XML AS BLOB( 10K ) xmlblob;
EXEC SQL END DECLARE SECTION;

sprintf( stmt, "XQUERY (10, xs:integer(1) to xs:integer(4))" );

EXEC SQL PREPARE s1 FROM :stmt;
EXEC SQL DECLARE c1 CURSOR FOR s1;
EXEC SQL OPEN c1;

while( sqlca.sqlcode == SQL_RC_OK )
{
  EXEC SQL FETCH c1 INTO :xmlblob;
  /* Mostrar resultados */
}

EXEC SQL CLOSE c1;
EXEC SQL COMMIT;
```

Ejemplo 2: Ejecución de expresiones XQuery en SQL estático utilizando la función XMLQUERY

Las sentencias de SQL que contienen la función XMLQUERY se pueden preparar de forma estática, del siguiente modo:

```
EXEC SQL INCLUDE SQLCA;
EXEC SQL BEGIN DECLARE SECTION;
  SQL TYPE IS XML AS BLOB( 10K ) xmlblob;
EXEC SQL END DECLARE SECTION;

EXEC SQL DECLARE C1 CURSOR FOR SELECT XMLQUERY( '(10, xs:integer(1) to
 xs:integer(4))' RETURNING SEQUENCE BY REF) from SYSIBM.SYSDUMMY1;

EXEC SQL OPEN c1;

while( sqlca.sqlcode == SQL_RC_OK )
{
  EXEC SQL FETCH c1 INTO :xmlblob;
  /* Mostrar resultados */
}

EXEC SQL CLOSE c1;
EXEC SQL COMMIT;
```

Ejemplo 3: Ejecución de expresiones XQuery en aplicaciones de SQL incorporado COBOL

En aplicaciones COBOL, las expresiones XQuery se pueden ejecutar del siguiente modo:

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
     01 stmt pic x(80).
   01 xmlBuff USAGE IS SQL TYPE IS XML AS BLOB (10K).
EXEC SQL END DECLARE SECTION END-EXEC.

MOVE "XQUERY (10, xs:integer(1) to xs:integer(4))" TO stmt.
EXEC SQL PREPARE s1 FROM :stmt END-EXEC.
EXEC SQL DECLARE c1 CURSOR FOR s1 END-EXEC.
EXEC SQL OPEN c1 USING :host-var END-EXEC.

*Llamar a FETCH y al bucle UPDATE.
Perform Fetch-Loop through End-Fetch-Loop
   until SQLCODE does not equal 0.

EXEC SQL CLOSE c1 END-EXEC.
EXEC SQL COMMIT END-EXEC.

Fetch-Loop Section.
    EXEC SQL FETCH c1 INTO :xmlBuff END-EXEC.
    if SQLCODE not equal 0
       go to End-Fetch-Loop.
* Mostrar resultados
End-Fetch-Loop. exit.
```

Ejecución de sentencias de SQL en aplicaciones de SQL incorporado

La ejecución de sentencias de SQL en aplicaciones de SQL incorporado es distinta para sentencias ejecutadas estáticamente y dinámicamente, aunque ambas utilizan el mandato EXEC SQL. Las sentencias estáticas están codificadas de manera fija en el código fuente de una aplicación de SQL incorporado. Las sentencias dinámicas se diferencian de las estáticas en que se compilan en tiempo de ejecución y pueden prepararse con parámetros de entrada. La información leída puede almacenarse en un medio denominado cursor, que entonces permite a los usuarios desplazarse libremente por los datos y realizar las actualizaciones correspondientes. La información de errores de SQLCODE, SQLSTATE y SQLWARN es una herramienta útil para la ayuda para resolver problemas de una aplicación.

Comentarios en aplicaciones de SQL incorporado

Los comentarios en cualquier aplicación son importantes para que el código de la aplicación se pueda entender. Esta sección trata sobre cómo añadir comentarios en aplicaciones de SQL incorporado.

Comentarios en aplicaciones de SQL incorporado C y C++

Cuando se trabaja con aplicaciones C y C++, se pueden insertar comentarios de SQL dentro del bloque EXEC SQL. Por ejemplo:

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

Comentarios en aplicaciones de SQL incorporado COBOL

Cuando se trabaja con aplicaciones COBOL, se pueden insertar comentarios de SQL dentro del bloque EXEC SQL. Por ejemplo:

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

Comentarios en aplicaciones de SQL incorporado FORTRAN

Cuando se trabaja con aplicaciones FORTRAN, se pueden insertar comentarios de SQL dentro del bloque EXEC SQL. Por ejemplo:

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

Comentarios en aplicaciones de SQL incorporado REXX

Los comentarios de SQL no están soportados en aplicaciones REXX.

Ejecución de sentencias de SQL estático en aplicaciones de SQL incorporado

Las sentencias de SQL se pueden ejecutar estáticamente en un lenguaje principal utilizando el siguiente enfoque:

- C o 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;
  ```

- 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 una tabla; `job-update` es una referencia a una variable del lenguaje principal declarada en la sección de declaración del código fuente:

```
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.
```

Recuperación de información de variables del lenguaje principal procedente de la estructura SQLDA en aplicaciones de SQL incorporado

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).

Cuando la sentencia DESCRIBE se ejecuta en la aplicación, el gestor de bases de datos define las variables del lenguaje principal en una SQLDA. Cuando las variables del lenguaje principal se han definido en la SQLDA, puede utilizar la sentencia FETCH para asignar valores a las variables del lenguaje principal, mediante un cursor.

Declaración de la estructura SQLDA en un programa de SQL ejecutado dinámicamente

La SQLDA contiene un número de ocurrencias de variables de entradas SQLVAR, cada una de las cuales contiene un grupo de campos que describen una columna de una fila de datos, tal como se muestra en la siguiente figura. Hay dos tipos de entradas SQLVAR: entradas SQLVAR base y entradas SQLVAR secundarias.

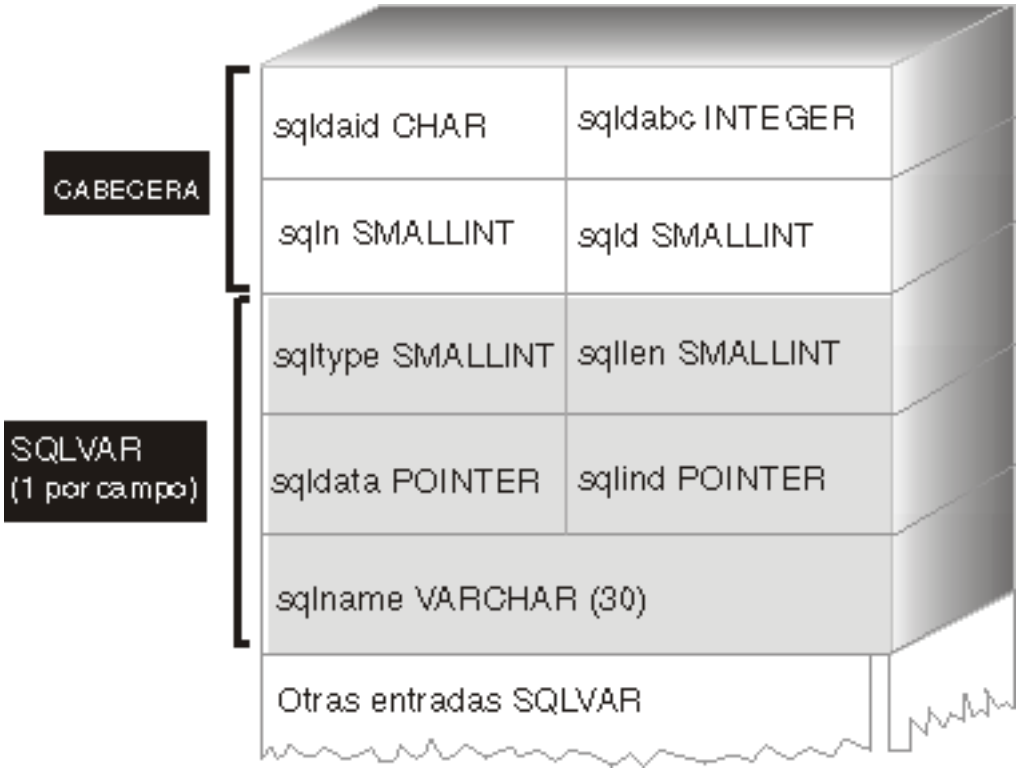


Figura 2. El Área de descriptores de SQL (SQLDA)

Puesto que el número de entradas SQLVAR necesario depende del número de columnas de la tabla de resultados, una aplicación debe ser capaz de asignar un número adecuado de elementos SQLVAR cuando se necesiten. Utilice uno de los siguientes métodos:

- Proporcione la SQLDA de mayor tamaño (es decir, la que tenga el mayor número de entradas SQLVAR) que se necesite. El número máximo de columnas que se pueden devolver en una tabla de resultados es 255. Si cualquiera de las columnas devueltas es un tipo LOB o un tipo diferenciado, el valor de SQLN se dobla y el número de entradas SQLVAR necesarias para albergar la información se dobla a 510. Sin embargo, puesto que la mayoría de sentencias SELECT no recuperan ni 255 columnas, la mayoría del espacio asignado no se utiliza.
- Proporcione una SQLDA de menor tamaño con menos entradas SQLVAR. En este caso, si hay más columnas en el resultado que entradas SQLVAR permitidas en la SQLDA, no se devuelve ninguna descripción. En su lugar, el gestor de bases de datos devuelve el número de elementos de lista de selección detectado en la sentencia SELECT. La aplicación asigna una SQLDA con el número de entradas SQLVAR necesario y utiliza la sentencia DESCRIBE para adquirir las descripciones de columnas.
- Cuando el número de columnas devuelto es del tipo LOB o definido por el usuario, proporcione una SQLDA con el número exacto de entradas SQLVAR.

Para los tres métodos, la duda es cuántas entradas SQLVAR iniciales se deben asignar. Cada elemento SQLVAR utiliza un máximo de 44 bytes de almacenamiento (sin contar el almacenamiento asignado para los campos SQLDATA y SQLIND). Si hay memoria suficiente, el primer método de proporcionar una SQLDA de tamaño máximo resulta más fácil de implementar.

El segundo método de asignar una SQLDA de menor tamaño sólo se aplica a lenguajes de programación, como C y C++, que dan soporte a la asignación dinámica de memoria. Para lenguajes como COBOL y FORTRAN, que no dan soporte a la asignación dinámica de memoria, utilice el primer método.

Preparación de una sentencia de SQL ejecutada dinámicamente utilizando la estructura SQLDA mínima

Utilice la información de este tema como ejemplo sobre cómo asignar la estructura SQLDA mínima para una sentencia.

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.

Supongamos que una aplicación declara una estructura SQLDA denominada `minsqlda` que no contiene ninguna entrada SQLVAR. El campo SQLN de la SQLDA describe el número de entradas SQLVAR que se asignan. En este caso, SQLN se debe establecer en 0. A continuación, para preparar una sentencia desde la serie de caracteres `dstring` y para entrar su descripción en `minsqlda`, emita la siguiente sentencia de SQL (suponiendo que se utiliza sintaxis de C y que `minsqlda` se declara como un puntero a una estructura SQLDA):

```
EXEC SQL
  PREPARE STMT INTO :*minsqlda FROM :dstring;
```

Supongamos que la sentencia contenida es `dstring` es una sentencia SELECT que devuelve 20 columnas en cada fila. Después de la sentencia PREPARE (o de una sentencia DESCRIBE), el campo SQLD de la SQLDA contiene el número de columnas de la tabla de resultados correspondiente a la sentencia SELECT preparada.

Las entradas SQLVAR de la SQLDA se establecen en los casos siguientes:

- SQLN >= SQLD y ninguna columna tiene el tipo LOB ni un tipo diferenciado.

  Las primeras entradas SQLVAR de SQLD se establecen y SQLDOUBLED se establece en blanco.
- SQLN >= 2*SQLD y al menos una columna es de tipo LOB o tiene un tipo diferenciado.

  2* entradas SQLVAR de SQLD se establecen y SQLDOUBLED se establece en 2.
- SQLD <= SQLN < 2*SQLD y al menos una columna tiene un tipo diferenciado, pero no hay ninguna columna LOB.

  Las primeras entradas SQLVAR de SQLD se establecen y SQLDOUBLED se establece en blanco. Si la opción de vinculación SQLWARN tiene el valor YES, se emite un aviso SQLCODE +237 (SQLSTATE 01594).

Las entradas SQLVAR de la SQLDA *no* se establecen (es necesaria la asignación de espacio adicional y otra sentencia DESCRIBE) en los casos siguientes:

- SQLN < SQLD y ninguna columna tiene el tipo LOB ni un tipo diferenciado.

  No se establece ninguna entrada SQLVAR y SQLDOUBLED se establece en blanco. Si la opción de vinculación SQLWARN tiene el valor YES, se emite un aviso SQLCODE +236 (SQLSTATE 01005).

  Asigne entradas SQLVAR de SQLD para lograr una operación DESCRIBE satisfactoria.
- SQLN < SQLD y al menos una columna tiene un tipo diferenciado, pero no hay ninguna columna LOB.

No se establece ninguna entrada SQLVAR y SQLDOUBLED se establece en blanco. Si la opción de vinculación SQLWARN tiene el valor YES, se emite un aviso SQLCODE +239 (SQLSTATE 01005).

Asigne 2*SQLD entradas SQLVAR para lograr una operación DESCRIBE satisfactoria, incluidos los nombres de los tipos diferenciados.

- SQLN < 2*SQLD y al menos una columna es de tipo LOB.

  No se establece ninguna entrada SQLVAR y SQLDOUBLED se establece en blanco. Se emite un aviso SQLCODE +238 (SQLSTATE 01005) (independientemente del valor de la opción de vinculación SQLWARN).

  Asigne 2*SQLD entradas SQLVAR para lograr una operación DESCRIBE satisfactoria.

La opción SQLWARN del mandato BIND sirve para controlar si DESCRIBE (o PREPARE...INTO) devolverá los siguientes avisos:

- SQLCODE +236 (SQLSTATE 01005)
- SQLCODE +237 (SQLSTATE 01594)
- SQLCODE +239 (SQLSTATE 01005).

Se recomienda que el código de la aplicación siempre tenga en cuenta que se pueden devolver estos valores SQLCODE. Siempre se devuelve el aviso SQLCODE +238 (SQLSTATE 01005) cuando hay columnas LOB en la lista de selección y hay un número insuficiente de entradas SQLVAR en la SQLDA. Esta es la única manera que tiene la aplicación para saber que el número de entradas SQLVAR se debe doblar debido a una columna LOB en el conjunto de resultados.

Asignación de una estructura SQLDA con suficientes entradas SQLVAR para sentencias de SQL ejecutadas dinámicamente

Después de determinar el número de columnas de la tabla de resultados, asigne almacenamiento para un segundo SQLDA de tamaño completo. El primer SQLDA se utiliza para parámetros de entrada y el segundo SQLDA de tamaño completo se utiliza para parámetros de salida.

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))
```

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.

Descripción de una sentencia SELECT en un programa de SQL ejecutado dinámicamente

Después de asignar espacio suficiente para el segundo SQLDA (en este ejemplo, denominado `fulsqlda`), debe codificar la aplicación para que describa la sentencia SELECT.

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 emitirá la sentencia siguiente:

```
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.

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.

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) porque la pila no puede asignar suficiente memoria. 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.

Proceso del cursor en un programa SQL ejecutado dinámicamente

Después de asignar correctamente la estructura SQLDA, el cursor asociado con la sentencia SELECT se puede abrir y se pueden captar filas.

Para procesar el cursor asociado con una sentencia SELECT, primero abra el cursor y luego capte filas especificando la cláusula USING DESCRIPTOR de la sentencia FETCH. Por ejemplo, una aplicación C podría tener lo siguiente:

```
EXEC SQL OPEN pcurs
EMB_SQL_CHECK( "OPEN" ) ;
EXEC SQL FETCH pcurs USING DESCRIPTOR :*sqldaPointer
EMB_SQL_CHECK( "FETCH" ) ;
```

Para procesar una operación FETCH satisfactoriamente, podría escribir la aplicación de modo que obtuviera los datos de la SQLDA y mostrara las cabeceras de columna. Por ejemplo:

```
display_col_titles( sqldaPointer ) ;
```

Una vez visualizados los datos, debería cerrar el cursor y liberar la memoria asignada de forma dinámica. Por ejemplo:

```
EXEC SQL CLOSE pcurs ;
EMB_SQL_CHECK( "CLOSE CURSOR" ) ;
```

Asignación de una estructura SQLDA para un programa de SQL ejecutado dinámicamente

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.

Para crear una estructura SQLDA con C, incorpore la sentencia INCLUDE SQLDA en el lenguaje principal o incluya el archivo de inclusión SQLDA para obtener la definición de estructura. Luego, debido a que el tamaño de una SQLDA no es fijo, la aplicación debe declarar un puntero a una estructura SQLDA y asignar almacenamiento para la misma. El tamaño real de la estructura SQLDA depende del número de elementos de datos diferenciados que se pasan mediante la SQLDA.

En el lenguaje de programación C y C++, se proporciona una macro que facilita la asignación de SQLDA. Esta macro tiene el siguiente formato:

```
#define SQLDASIZE(n) (offsetof(struct sqlda, sqlvar) \
        + (n) × sizeof(struct sqlvar))
```

El efecto de esta macro es calcular el almacenamiento necesario para una SQLDA con `n` elementos SQLVAR.

Para crear una estructura SQLDA con COBOL, puede incorporar una sentencia INCLUDE SQLDA o utilizar la sentencia COPY. Utilice la sentencia COPY cuando desee controlar el número máximo de entradas SQLVAR y por tanto la cantidad de espacio de almacenamiento ocupado por la SQLDA. Por ejemplo, para cambiar el número por omisión de entradas SQLVAR de 1489 a 1, utilice la siguiente sentencia COPY:

```
COPY "sqlda.cbl"
  replacing --1489--
  by --1--.
```

El lenguaje FORTRAN no da soporte directamente a las estructuras de datos autodefinidas ni a la asignación dinámica. No se proporciona ningún archivo de inclusión SQLDA para FORTRAN porque no es posible dar soporte a la SQLDA como una estructura de datos en FORTRAN. El precompilador pasará por alto la sentencia INCLUDE SQLDA en un programa FORTRAN.

Sin embargo, puede crear algo parecido a una estructura SQLDA estática en un programa FORTRAN y utilizar dicha estructura siempre que se pueda utilizar una SQLDA. El archivo `sqldact.f` contiene constantes que ayudan a declarar una estructura SQLDA en FORTRAN.

Ejecute llamadas a SQLGADDR para asignar valores de puntero a elementos de SQLDA que los necesiten.

La tabla siguiente muestra la declaración y uso de una estructura SQLDA con un elemento SQLVAR.

Lenguaje	Código fuente de ejemplo

C y C++	`#include`
	`struct sqlda *outda = (struct sqlda *)malloc(SQLDASIZE(1));`
	
	`/* DECLARAR VARIABLES LOCALES PARA ALBERGAR DATOS REALES */`
	`double sal = 0;`
	`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;`

Lenguaje	Código fuente de ejemplo

COBOL

```
 WORKING-STORAGE SECTION.
 77 SALARY          PIC S99999V99 COMP-3.
 77 SAL-IND         PIC S9(4)     COMP-5.

    EXEC SQL INCLUDE SQLDA END-EXEC

* O codificar un modo útil para guardar entradas SQLVAR no utilizadas.
* COPY "sqlda.cbl" REPLACING --1489-- BY --1--.

      01 decimal-sqllen pic s9(4) comp-5.
      01 decimal-parts redefines decimal-sqllen.
         05 precision pic x.
         05 scale pic x.

* Inicializar un elemento de SQLDA de salida
      MOVE 1 TO SQLN
      MOVE 1 TO SQLD
      MOVE SQL-TYP-NDECIMAL TO SQLTYPE(1)

* Longitud = 7 dígitos de precisión y 2 dígitos de escala

      MOVE x"07" TO PRECISION.
      MOVE x"02" TO SCALE.
      MOVE DECIMAL-SQLLEN TO O-SQLLEN(1).
      SET SQLDATA(1) TO ADDRESS OF SALARY
      SET SQLIND(1)  TO ADDRESS OF SAL-IND
```

Lenguaje	Código fuente de ejemplo

FORTRAN

```
      include 'sqldact.f'

      integer*2  sqlvar1
      parameter ( sqlvar1 = sqlda_header_sz + 0*sqlvar_struct_sz )

C     Declarar una SQLDA de salida -- 1 Variable
      character    out_sqlda(sqlda_header_sz + 1*sqlvar_struct_sz)

      character*8  out_sqldaid     ! Cabecera
      integer*4    out_sqldabc
      integer*2    out_sqln
      integer*2    out_sqld

      integer*2    out_sqltype1    ! Primera variable
      integer*2    out_sqllen1
      integer*4    out_sqldata1
      integer*4    out_sqlind1
      integer*2    out_sqlnamel1
      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_sqlnamel1                                   )
      equivalence( out_sqlda(sqlvar1+sqlvar_name_data_ofs),
     +             out_sqlnamec1                                   )

C     Declarar variables locales para albergar los datos devueltos.
      real*8       salary
      integer*2    sal_ind

C     Inicializar la SQLDA de salida (cabecera)
      out_sqldaid  = 'OUT_SQLDA'
      out_sqldabc  = sqlda_header_sz + 1*sqlvar_struct_sz
      out_sqln     = 1
      out_sqld     = 1
C     Inicializar VAR1
      out_sqltype1 = SQL_TYP_NFLOAT
      out_sqllen1  = 8
      rc = sqlgaddr( %ref(salary), %ref(out_sqldata1) )
      rc = sqlgaddr( %ref(sal_ind), %ref(out_sqlind1) )
```

Nota: El ejemplo anterior se ha escrito para FORTRAN de 32 bits.

En lenguajes que no dan soporte a la asignación dinámica de memoria, se debe declarar de forma explícita en el lenguaje principal una SQLDA con el número deseado de elementos SQLVAR. Asegúrese de declarar el número suficiente de elementos SQLVAR según los requisitos de la aplicación.

Transferencia de datos en un programa de SQL ejecutado dinámicamente utilizando una estructura SQLDA

Puede obtener una mayor flexibilidad si transfiere datos mediante una SQLDA que si utiliza listas de variables del lenguaje principal. Por ejemplo, puede utilizar una

SQLDA para transferir datos que no tienen ningún equivalente en el lenguaje principal, como datos DECIMAL en lenguaje C.

Utilice la siguiente tabla como listado de referencias cruzadas que muestra cómo se relacionan los valores numéricos y los nombres simbólicos.

Tabla 17. 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/a[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/a[3]	460/461	SQL_TYP_CSTR / SQL_TYP_NCSTR
VARGRAPHIC	464/465	SQL_TYP_VARGRAPH / SQL_TYP_NVARGRAPH
GRAPHIC	468/469	SQL_TYP_GRAPHIC / SQL_TYP_NGRAPHIC
LONG VARGRAPHIC	472/473	SQL_TYP_LONGRAPH / SQL_TYP_NLONGRAPH
FLOAT	480/481	SQL_TYP_FLOAT / SQL_TYP_NFLOAT
REAL[4]	480/481	SQL_TYP_FLOAT / SQL_TYP_NFLOAT
DECIMAL[5]	484/485	SQL_TYP_DECIMAL / SQL_TYP_DECIMAL
INTEGER	496/497	SQL_TYP_INTEGER / SQL_TYP_NINTEGER
SMALLINT	500/501	SQL_TYP_SMALL / SQL_TYP_NSMALL
n/a	804/805	SQL_TYP_BLOB_FILE / SQL_TYPE_NBLOB_FILE
n/a	808/809	SQL_TYP_CLOB_FILE / SQL_TYPE_NCLOB_FILE
n/a	812/813	SQL_TYP_DBCLOB_FILE / SQL_TYPE_NDBCLOB_FILE
n/a	960/961	SQL_TYP_BLOB_LOCATOR / SQL_TYP_NBLOB_LOCATOR
n/a	964/965	SQL_TYP_CLOB_LOCATOR / SQL_TYP_NCLOB_LOCATOR
n/a	968/969	SQL_TYP_DBCLOB_LOCATOR / SQL_TYP_NDBCLOB_LOCATOR
XML	988/989	SQL_TYP_XML / SQL_TYP_XML

Nota: Estos tipos definidos se encuentran en el archivo de inclusión `sql.h` del subdirectorio `include` del directorio `sqllib`. (Por ejemplo, `sqllib/include/sql.h` para el lenguaje de programación C.)

1. Para el lenguaje de programación COBOL, el nombre SQLTYPE no utiliza el signo de subrayado (_) sino un guión (-).
2. Es una serie gráfica terminada en nulo.
3. Es una serie de caracteres terminada en nulo.
4. La diferencia entre REAL y DOUBLE en la SQLDA es el valor de la longitud (4 u 8).
5. La precisión está en el primer byte. La escala está en el segundo byte.

Proceso de sentencias de SQL interactivas en programas de SQL ejecutados dinámicamente

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. Los valores desconocidos hasta el momento de la ejecución se pueden representar mediante marcadores de parámetro, que se indican mediante signos de interrogación. Los marcadores de parámetro permiten la interacción entre el usuario y la aplicación y se parecen a las variables del lenguaje principal para sentencias de SQL estático.

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.

Determinación del tipo de sentencia en programas de SQL ejecutados dinámicamente

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.

Proceso de sentencias SELECT de lista de variables en programas de SQL ejecutados dinámicamente

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.

Para procesar una sentencia SELECT de lista de variables, codifique la aplicación de modo que haga lo siguiente:

1. Declare una SQLDA.

   Se debe utilizar una estructura SQLDA para procesar las sentencias SELECT de lista de variables.
2. PREPARE la sentencia mediante la cláusula INTO.

   Luego la aplicación determina si la estructura SQLDA declarada tiene suficientes elementos SQLVAR. Si no es así, la aplicación asigna otra estructura

SQLDA con el número necesario de elementos SQLVAR y emite una sentencia DESCRIBE adicional mediante el nuevo SQLDA.

3. Asigne los elementos SQLVAR.

   Asigne almacenamiento para las variables del lenguaje principal e indicadores necesarios para cada SQLVAR. Este paso incluye la colocación de las direcciones asignadas para los datos y variables de indicador en cada elemento SQLVAR.

4. Procese la sentencia SELECT.

   Se asocia un cursor con la sentencia preparada, se abre y las filas se captan mediante la estructura SQLDA correctamente asignada.

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.

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.

Cómo proporcionar entrada de variables a una sentencia de SQL ejecutada dinámicamente 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.

Supongamos que la aplicación utiliza SQL dinámico y que desea que pueda realizar una operación DELETE. Una serie de caracteres que contenga un marcador de parámetros puede tener un aspecto parecido al siguiente:

```
DELETE FROM TEMPL WHERE EMPNO = ?
```

Cuando se ejecuta esta sentencia, se especifica una variable del lenguaje principal o una estructura SQLDA mediante la cláusula USING de la sentencia EXECUTE. El contenido de la variable del lenguaje principal se utiliza cuando se ejecuta la sentencia.

El marcador de parámetros adopta un tipo de datos y longitud supuestos que dependen del contexto de su uso dentro de la sentencia de SQL. Si el tipo de datos de un marcador de parámetros no resulta obvio a partir del contexto de la sentencia en la que se utiliza, utilice CAST para especificar el tipo. Dicho marcador de parámetros se considera un a *marcador de parámetros tipificado*. Los marcadores de parámetros tipificados se tratan como una variable del lenguaje principal del tipo determinado. Por ejemplo, la sentencia `SELECT ? FROM SYSCAT.TABLES` no es válida porque DB2 no conoce el tipo de la columna de resultados. Sin embargo, la sentencia `SELECT CAST(? AS INTEGER) FROM SYSCAT.TABLES` es válida porque cast indica que el marcador de parámetros representa un INTEGER, de modo que DB2 conoce el tipo de la columna de resultados.

Si la sentencia de SQL contiene más de un marcador de parámetros, la cláusula USING de la sentencia EXECUTE debe especificar una lista de variables del lenguaje principal (una para cada marcador de parámetros) o debe identificar una SQLDA que tenga una entrada SQLVAR para cada marcador de parámetros. (Observe que para los LOB, existen dos entradas SQLVAR por cada marcador de parámetros.) La lista de variables del lenguaje principal o entradas SQLVAR se comparan de acuerdo con el orden de los marcadores de parámetros en la sentencia y deben tener tipos de datos compatibles.

Nota: El utilizar un marcador de parámetros con SQL dinámico es como utilizar variables del lenguaje principal con SQL estático. En cualquier caso, el optimizador no utiliza estadísticas de distribución y es posible que no elija el mejor plan de acceso.

Las normas que se aplican a marcadores de parámetros se describen con la sentencia PREPARE.

Ejemplo de marcadores de parámetros en un programa de SQL ejecutado dinámicamente

Los siguientes ejemplos muestran cómo utilizar marcadores de parámetros en un programa de SQL dinámico:

- C y 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 parámetros */
strcpy(hostVarStmt1, "DELETE FROM org WHERE deptnumb = ?");
EXEC SQL PREPARE Stmt1 FROM :hostVarStmt1;

/* ejecutar la sentencia correspondiente a hostVarDeptnumb = 15 */
hostVarDeptnumb = 15;
EXEC SQL EXECUTE Stmt1 USING :hostVarDeptnumb;
```

- COBOL (**varinp.sqb**)

  El siguiente ejemplo procede del ejemplo de COBOL **varinp.sqb** y muestra cómo utilizar un marcador de parámetros en condiciones de búsqueda y actualización:

```
   EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 pname             pic x(10).
01 dept              pic s9(4) comp-5.
01 st                pic x(127).
01 parm-var          pic x(5).
```

```
   EXEC SQL END DECLARE SECTION END-EXEC.

   move "SELECT name, dept FROM staff
-         "   WHERE job = ? FOR UPDATE OF job" to st.
   EXEC SQL PREPARE s1 FROM :st END-EXEC.

   EXEC SQL DECLARE c1 CURSOR FOR s1 END-EXEC.

   move "Mgr" to parm-var.
   EXEC SQL OPEN c1 USING :parm-var END-EXEC

   move "Clerk" to parm-var.
   move "UPDATE staff SET job = ? WHERE CURRENT OF c1" to st.
   EXEC SQL PREPARE s2 from :st END-EXEC.

* llamar a FETCH y al bucle UPDATE.
   perform Fetch-Loop thru End-Fetch-Loop
      until SQLCODE not equal 0.

   EXEC SQL CLOSE c1 END-EXEC.
```

Llamada a procedimientos almacenados en aplicaciones de SQL incorporado

Se puede llamar a procedimientos almacenados desde aplicaciones de SQL incorporado formulando y ejecutando la sentencia CALL con los parámetros y referencia a procedimiento adecuados. La sentencia CALL se puede ejecutar de forma estática o dinámica dentro de aplicaciones de SQL incorporado. Sin embargo, para cada lenguaje de programación hay distintos métodos para ejecutar este mandato. Independientemente del lenguaje principal, cada variable del lenguaje principal utilizada en el procedimiento almacenado se tiene que declarar de modo que coincida con el tipo de datos necesario.

Las aplicaciones cliente y la llamada de rutinas intercambian información con los procedimientos a través de parámetros y conjuntos de resultados. Los parámetros correspondientes a procedimientos están definidos por la dirección en que viajan los datos (la modalidad de parámetro).

Existen tres tipos de parámetros para los procedimientos:

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

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

Llamada a procedimientos almacenados en aplicaciones de SQL incorporado C y C++

Llamada a procedimientos almacenados en aplicaciones de SQL incorporado C y C++

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

Cuando se trabaja con aplicaciones C y C++, se puede llamar a un procedimiento almacenado, INOUT_PARAM, mediante la siguiente sentencia:

```
EXEC SQL CALL INOUT_PARAM(:inout_median:medianind, :out_sqlcode:codeind,
                          :out_buffer:bufferind);
```

donde inout_median, out_sqlcode y out_buffer son variables del lenguaje principal y medianind, codeind y bufferind son variables de indicadores de nulo.

Nota: También se puede llamar a procedimientos almacenados dinámicamente preparando una sentencia CALL.

Llamada a procedimientos almacenados desde REXX

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.)

Lectura de los conjuntos de resultados y desplazamiento por los mismos en aplicaciones de SQL incorporado

Una de las tareas más comunes de un programa de aplicación de SQL incorporado 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.

Nota: Las aplicaciones de SQL incorporado pueden llamar a procedimientos almacenados con cualquiera de las implementaciones de procedimientos almacenados soportadas y pueden recuperar valores de parámetros de salida y de entrada-salida; sin embargo, las aplicaciones de SQL incorporado no pueden leer los conjuntos de resultados devueltos por procedimientos almacenados ni desplazarse por los mismos.

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

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

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

Desplazamiento por datos anteriormente recuperados en aplicaciones de SQL incorporado

Cuando una aplicación recupera datos de la base de datos, la sentencia FETCH le permite desplazarse hacia adelante por los datos; sin embargo, no hay ninguna sentencia de SQL que le permita desplazarse hacia atrás por el conjunto de resultados (equivalente a una operación FETCH hacia atrás). Sin embargo, la CLI

de DB2 y el controlador DB2 Universal JDBC dan soporte a la operación FETCH hacia atrás mediante cursores desplazables de sólo lectura.

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 en la memoria de la aplicación y desplácese por los mismos mediante alguna técnica de programación.
- Utilice SQL para recuperar de nuevo los datos, normalmente mediante una segunda sentencia SELECT.

Mantenimiento de una copia de datos captados aplicaciones de SQL incorporado

En algunas situaciones, puede resultar útil mantener una copia de los datos que capta la aplicación.

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.

Recuperación de datos captados por segunda vez en aplicaciones de SQL incorporado

La técnica que utilice para recuperar datos por segunda vez dependerá del orden en el que desee volver a ver los datos.

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.

Diferencias en el orden de las filas en tablas de resultados

Las filas de varias tablas correspondientes a la misma sentencia SELECT pueden no visualizarse en el mismo orden. 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 del 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.

Actualización de datos anteriormente recuperados en aplicaciones de SQL incorporado

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.

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.

Selección de varias filas utilizando un cursor en aplicaciones de SQL incorporado

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.

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.

Actualización y supresión de datos recuperados en una aplicación de SQL ejecutada estáticamente

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.

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.

Ejemplo de captación en un programa de SQL ejecutado estáticamente

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 y 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.

- 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.
```

Recuperación de mensajes de error en aplicaciones de SQL incorporado

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.

  Ejemplo de C: el procedimiento SqlInfoPrint de UTILAPI.C

```
/******************************************************************************
** 1.1 - SqlInfoPrint - imprime información de diagnóstico en la pantalla.
**
******************************************************************************/
int SqlInfoPrint( char * appMsg,
    struct sqlca * pSqlca,
    int line,
    char * file )
{   int  rc = 0;
    char sqlInfo[1024];
    char sqlInfoToken[1024];
    char sqlstateMsg[1024];
    char errorMsg[1024];
    if (pSqlca->sqlcode != 0 && pSqlca->sqlcode != 100)
    {  strcpy(sqlInfo, "");
       if( pSqlca->sqlcode < 0)
       {   sprintf( sqlInfoToken, "\n---- error report ----\n");
           strcat( sqlInfo, sqlInfoToken);
       }
       else
       {   sprintf( sqlInfoToken, "\n---- warning report ----\n");
           strcat( sqlInfo, sqlInfoToken);
       } /* endif */

       sprintf( sqlInfoToken, " app. message =
       strcat( sqlInfo, sqlInfoToken);
       sprintf( sqlInfoToken, " line =
       strcat( sqlInfo, sqlInfoToken);
       sprintf( sqlInfoToken, " file =
       strcat( sqlInfo, sqlInfoToken);
       sprintf( sqlInfoToken, " SQLCODE = sqlcode);
       strcat( sqlInfo, sqlInfoToken);

       /* obtener mensaje de error */
       rc = sqlaintp( errorMsg, 1024, 80, pSqlca);
       /* código de retorno es la longitud de la serie errorMsg */
       if( rc > 0)
       { sprintf( sqlInfoToken, "
         strcat( sqlInfo, sqlInfoToken);
       }

       /* obtener mensaje SQLSTATE */
       rc = sqlogstt( sqlstateMsg, 1024, 80, pSqlca->sqlstate);
       if (rc == 0)
       { sprintf( sqlInfoToken, "
         strcat( sqlInfo, sqlInfoToken);
       }

       if( pSqlca->sqlcode < 0)
       { sprintf( sqlInfoToken, "--- end error report ---\n");
         strcat( sqlInfo, sqlInfoToken);
```

```
          printf("
          return 1;
        }
        else
        { sprintf( sqlInfoToken, "--- fin informe avisos ---\n");
          strcat( sqlInfo, sqlInfoToken);

          printf("
          return 0;
        } /* endif */
    } /* endif */
    return 0;
}
```

```
Ejemplo en COBOL: de CHECKERR.CBL
    *******************************
    * Llamada a API GET ERROR MESSAGE *
    *******************************
      call "sqlgintp" using
           by value buffer-size
           by value line-width
           by reference sqlca
           by reference error-buffer
         returning error-rc.
    ************************
    * GET SQLSTATE MESSAGE *
    ************************
      call "sqlggstt" using
           by value buffer-size
           by value line-width
           by reference sqlstate
           by reference state-buffer
        returning state-rc.
    if error-rc is greater than 0
      display error-buffer.

    if state-rc is greater than 0
      display state-buffer.

    if state-rc is less than 0
      display "código de retorno de GET SQLSTATE =" state-rc.

    if SQLCODE is less than 0
      display "--- fin informe errores ---"
      go to End-Prog.

    display "--- fin informe errores ---"
    display "CONTINUANDO PROGRAMA CON AVISOS".
```

- Las aplicaciones REXX utilizan el procedimiento CHECKERR.

```
/******   CHECKERR - Comprobar SQLCODE *****/
CHECKERR:
  arg errloc

  if  ( SQLCA.SQLCODE = 0 ) then
    return 0
  else do
    say '--- informe de errores ---'
    say 'Se ha producido un ERROR:' errloc
    say 'SQLCODE :' SQLCA.SQLCODE

   /*********************\
   * GET ERROR MESSAGE  *
   \*********************/
    call SQLDBS 'GET MESSAGE INTO :errmsg LINEWIDTH 80'
    say errmsg
    say '--- fin informe errores ---'
```

```
      if (SQLCA.SQLCODE < 0 ) then
        exit
      else do
        say 'AVISO - CONTINUANDO PROGRAMA CON ERRORES'
        return 0
      end
    end
  return 0
```

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 de inclusión SQLCA. Los archivos fuente sin sentencias de SQL incorporado, pero que llaman a las API del gestor de bases de datos, también pueden proporcionar una o más estructuras SQLCA, pero sus nombres son arbitrarios.

Si la aplicación cumple con el estándar FIPS 127-2, puede declarar SQLSTATE y SQLCODE como variables del lenguaje principal para aplicaciones C, C++, COBOL y FORTRAN, en lugar de utilizar la estructura SQLCA.

Un valor SQLCODE igual a 0 significa una ejecución satisfactoria (con posibles condiciones de aviso SQLWARN). Un valor positivo significa que la sentencia se ha ejecutado satisfactoriamente pero con un aviso, como el truncamiento de una variable del lenguaje principal. Un valor negativo significa que se ha producido una condición de error.

Un campo adicional, SQLSTATE, contiene un código de error estandarizado coherente con otros productos de bases de datos de IBM y con otros gestores de bases de datos que cumplen con SQL92. En la práctica, es conveniente utilizar valores SQLSTATE cuando le preocupe la portabilidad, pues los valores SQLSTATE son utilizados por muchos gestores de bases de datos.

El campo SQLWARN contiene una matriz de indicadores de aviso, incluso si SQLCODE es cero. El primer elemento de la matriz SQLWARN, SQLWARN0, contiene un blanco si todos los demás elementos están en blanco. SQLWARN0 contiene una `W` si al menos uno de los otros elementos contiene un carácter de aviso.

Nota: Si desea desarrollar aplicaciones que acceden a varios servidores RDBMS de IBM, debe:

- Si es posible, hacer que las aplicaciones comprueben SQLSTATE en lugar de SQLCODE.
- Si la aplicación va a utilizar DB2 Connect, considerar la posibilidad de utilizar el recurso de correlación que proporciona DB2 Connect para correlacionar conversiones de SQLCODE entre bases de datos distintas.

Consideraciones sobre las rutinas de lista de salida

No utilice SQL ni llamadas a API de DB2 en rutinas de lista de salida. Tenga en cuenta que no puede desconectarse de una base de datos en una rutina de salida.

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 operativos 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.

Consulte la documentación de su plataforma para ver detalles específicos sobre consideraciones sobre distintos manejadores.

Desconexión de aplicaciones de SQL incorporado

La sentencia disconnect es el último paso cuando se trabaja con una base de datos. Este tema contiene ejemplos de la sentencia disconnect en los lenguajes principales soportados.

Desconexión de bases de datos DB2 en aplicaciones de SQL incorporado C y C++

Cuando se trabaja con aplicaciones C y C++, una conexión de base de datos se cierra emitiendo la siguiente sentencia:

```
EXEC SQL CONNECT RESET;
```

Desconexión de bases de datos DB2 en aplicaciones de SQL incorporado COBOL

Cuando se trabaja con aplicaciones COBOL, una conexión de base de datos se cierra emitiendo la siguiente sentencia:

```
EXEC SQL CONNECT RESET END-EXEC.
```

Desconexión de bases de datos DB2 en aplicaciones de SQL incorporado REXX

Cuando se trabaja con aplicaciones REXX, una conexión de base de datos se cierra emitiendo la siguiente sentencia:

```
CALL SQLEXEC 'CONNECT RESET'
```

Cuando se trabaja con aplicaciones FORTRAN, una conexión de base de datos se cierra emitiendo la siguiente sentencia:

```
EXEC SQL CONNECT RESET
```

Capítulo 6. Creación de aplicaciones de SQL incorporado

Cuando haya creado el código fuente para la aplicación de SQL incorporado, debe seguir otros pasos para construirla. Deberá considerar construir ejecutables de 64 bits al desarrollar nuevas aplicaciones de bases de datos de SQL incorporado. Además de compilar y enlazar el programa, debe *precompilarlo* y *vincularlo*.

El proceso de precompilación convierte sentencias de SQL incorporado en llamadas a API de tiempo de ejecución de DB2 que un compilador del lenguaje principal pueda procesar. Por omisión, se crea un paquete en el momento de la precompilación. Opcionalmente, se puede crear un archivo de vinculación en el momento de la precompilación. El archivo de vinculación contiene información sobre las sentencias de SQL del programa de aplicación. El archivo de vinculación se puede utilizar más adelante con el mandato `BIND` para crear un paquete para la aplicación.

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

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

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 para convertir las sentencias fuente de SQL incorporado en un formato que el gestor de bases de datos pueda utilizar

   Puesto que las sentencias de SQL colocadas 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 principal. Para los lenguajes C, C++, COBOL o FORTRAN, esta conversión se maneja mediante el precompilador de DB2 que se invoca mediante el mandato `PRECOMPILE` (o `PREP`). El precompilador convierte las sentencias de SQL incorporado directamente en llamadas a API de DB2 de servicios de tiempo de ejecución. Cuando el precompilador procesa un archivo fuente, busca específicamente sentencias de SQL y evita el lenguaje principal que no es SQL.
3. Compile los archivos fuente modificados (y otros archivos sin sentencias de SQL) mediante el compilador del lenguaje principal.
4. Enlace los archivos de objeto con las bibliotecas de DB2 y del lenguaje principal para generar un programa ejecutable.

   Compilación y enlace (pasos 3 y 4), para crear los módulos de objeto necesarios
5. Vincule el archivo de vinculación para crear el paquete si esto no se ha hecho en el momento de la precompilación o si se va a acceder a una base de datos distinta. La vinculación crea el paquete que utilizará el gestor de bases de datos cuando se ejecute el programa.
6. Ejecute la aplicación. La aplicación accede a la base de datos utilizando los planes de acceso.

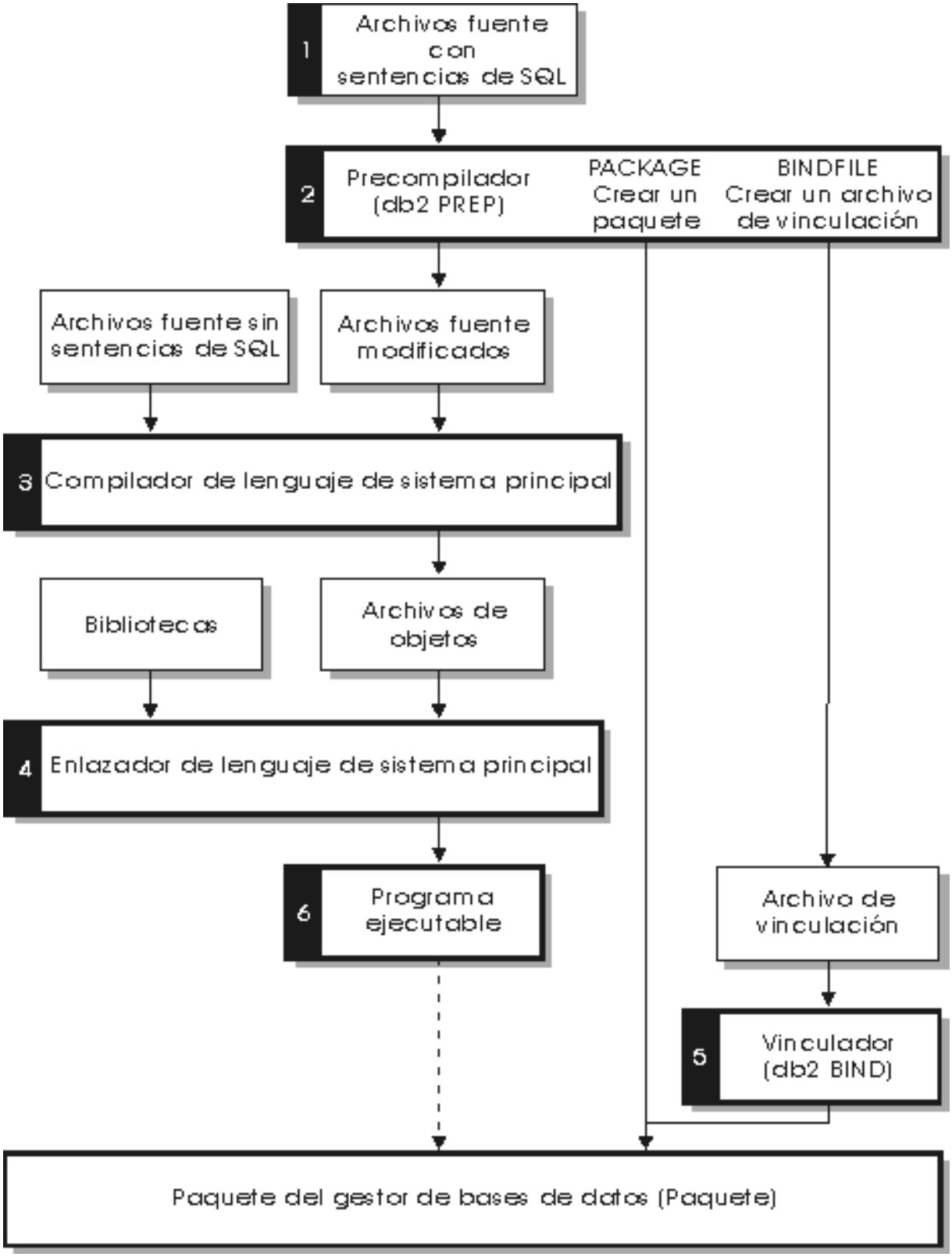


Figura 3. Preparación de programas escritos en lenguajes principal compilados

Precompilación de aplicaciones de SQL incorporado con el mandato PRECOMPILE

Cuando haya creado los archivos fuente de la aplicación de SQL incorporado, debe precompilar cada archivo del lenguaje principal que contenga sentencias de SQL con el mandato PREP, utilizando las opciones específicas del 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.

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*.

Nota: No está soportado ejecutar una aplicación incorporada en una versión de cliente anterior a la del cliente en el que se ha realizado la precompilación, independientemente de dónde se ha compilado la aplicación. Por ejemplo, no está soportado precompilar una aplicación incorporada en un cliente de DB2 V9.5 y luego intentar ejecutar la aplicación en un cliente DB2 V9.1.
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.

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.

Precompilación de aplicaciones de SQL incorporado que acceden a más de un servidor de bases de datos

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 a través de DB2 Connect. Precompílela contra el servidor al que se va a conectar, utilizando las opciones de `PREP` disponibles para dicho servidor.

Paquetes de aplicaciones y planes de acceso de SQL incorporado

El precompilador genera un paquete en la base de datos y, opcionalmente, un archivo de vinculación, si especifica que desea que se cree uno.

El paquete 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.

Los paquetes almacenados en la base de datos incluyen la 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.

El archivo de vinculación contiene las sentencias de SQL y otros datos necesarios para crear un paquete. Puede utilizar el archivo de vinculación para revincular la aplicación más adelante sin tener que precompilarla primero. La revinculación crea paquetes 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.

Calificación de esquema de paquete utilizando el registro especial CURRENT PACKAGE PATH

Los esquemas de paquete proporcionan un método para la agrupación lógica de paquetes. Existen diferentes métodos para agrupar paquetes y formar esquemas. Algunas implementaciones utilizan un esquema por cada entorno (por ejemplo, un esquema de producción y un esquema de prueba). Otras implementaciones utilizan un esquema por área de negocio (por ejemplo, los esquemas `stocktrd` y `onlinebnk`) o un esquema por aplicación (por ejemplo, `stocktrdAddUser` y `onlinebnkAddUser`).

También puede agrupar paquetes con finalidades de administración general o para proporcionar variaciones en los paquetes (por ejemplo, mantener variaciones de copia de seguridad de aplicaciones o probar variaciones nuevas de aplicaciones).

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

Nota: Únicamente DB2 Versión 9.1 para z/OS (DB2 para z/OS) tiene un registro especial de CURRENT PACKAGESET, que le permitirá definir explícitamente el valor (un único nombre de esquema) con la sentencia SET CURRENT PACKAGESET correspondiente. Aunque DB2 Database para Linux, UNIX y Windows tiene una sentencia SET CURRENT PACKAGESET, no tiene un registro especial CURRENT PACKAGESET. Esto significa que no se puede hacer referencia a CURRENT PACKAGESET en otros contextos (por ejemplo en una sentencia (por ejemplo en una sentencia SELECT) con DB2 Database para Linux, UNIX y Windows. DB2 para i5/OS no proporciona soporte para CURRENT PACKAGESET.

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

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

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

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

Muchas instalaciones utilizan más de un esquema para los paquetes. Si no especifica una lista de esquemas de paquete, deberá emitir la sentencia SET CURRENT PACKAGESET (que puede contener como máximo un nombre de esquema) cada vez que necesite un paquete de un esquema diferente. Sin embargo, si emite una sentencia SET CURRENT PACKAGE PATH al principio de la aplicación para especificar una lista de nombres de esquema, no necesita emitir una sentencia SET CURRENT PACKAGESET cada vez que se necesite un paquete en un esquema diferente.

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

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

Nota: En DB2 Database para Linux, UNIX y Windows, podrá definir el registro especial CURRENT PACKAGE PATH en el archivo db2cli.ini, utilizando la API de SQLSetConnectAttr, en la estructura de SQLE-CLIENT-INFO e incluyendo la sentencia SET CURRENT PACKAGE PATH en programas SQL incorporados. Sólo DB2 para z/OS, Versión 8 o posterior, da soporte a la sentencia SET CURRENT PACKAGE PATH. Si emite esta sentencia en un servidor de DB2 Database para Linux, UNIX y Windows o en DB2 para i5/OS, se devuelve -30005.

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

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

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

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

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

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

Nota: El proceso de precompilación de DB2 para z/OS almacena una señal de coherencia en el DBRM (que puede definirse utilizando la opción LEVEL) y durante la resolución de paquetes, se efectúa una comprobación para asegurarse de que la señal de coherencia del programa corresponde al paquete. De modo análogo, el proceso de vinculación de DB2 Database para Linux, UNIX y Windows almacena una indicación horaria en el proceso de vinculación. DB2 Database para Linux, UNIX y Windows también da soporte a una opción LEVEL.

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

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

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

Indicaciones horarias generadas por el precompilador

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

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

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

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

Errores y avisos procedentes de la precompilación aplicaciones de SQL incorporado

El precompilador de SQL incorporado detecta los errores de SQL incorporado en el momento de la precompilación. El precompilador de SQL incorporado detecta los errores de sintaxis, por ejemplo la falta de signos de punto y coma y las variables del lenguaje principal no declaradas en sentencias de SQL. Para cada uno de estos errores, se genera un mensaje de error adecuado.

Compilación y enlace de archivos fuente que contienen SQL incorporado

Cuando se precompilan archivos fuente de SQL incorporado, el mandato `PRECOMPILE` genera archivos fuente modificados con una extensión de archivo que se puede aplicar al lenguaje de programación.

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 operativos basados en UNIX y Linux, la aplicación puede ser un módulo de carga ejecutable o una biblioteca compartida.

Nota: Aunque las aplicaciones pueden ser bibliotecas de enlace dinámico (DLL) en los sistemas operativos Windows, las DLL las carga directamente la aplicación y no gestor de bases de datos de DB2. En sistemas operativos Windows, el gestor de bases de datos carga los procedimientos almacenados de SQL incorporado y las funciones definidas por el usuario como DLL.

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.

Vinculación de paquetes de SQL incorporado con una base de datos

La vinculación es el proceso de crear un paquete a partir de un archivo de vinculación y de almacenarlo en una base de datos.

Relaciones entre aplicación, archivo de vinculación y paquete

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

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.

Efecto de la opción de vinculación DYNAMICRULES en SQL dinámico

La opción DYNAMICRULES de los mandatos PRECOMPILE y BIND determina los valores que se aplicarán en el momento de la ejecución para los siguientes atributos de SQL:

- El ID de autorización que se utiliza durante la comprobación de autorización.
- El calificador que se utiliza para la calificación de objetos no calificados.
- Si el paquete se puede utilizar para preparar de forma dinámica las siguientes sentencias: GRANT, REVOKE, ALTER, CREATE, DROP, COMMENT ON, RENAME, SET INTEGRITY y SET EVENT MONITOR STATE.

Además del valor de DYNAMICRULES, el entorno de tiempo de ejecución de un paquete controla el modo en que se comportan las sentencias de SQL en el momento de la ejecución. Los dos entornos de ejecución posibles son:

- El paquete se ejecuta formando parte de un programa autónomo
- El paquete se ejecuta dentro del contexto de una rutina

La combinación del valor de DYNAMICRULES y del entorno de tiempo de ejecución determina los valores correspondientes a los atributos de SQL dinámico. Este conjunto de valores de atributos se denomina comportamiento de las sentencias de SQL dinámico. Los cuatro comportamientos son:

Comportamiento de ejecución
DB2 utiliza el ID de autorización del usuario (el ID que se conectó inicialmente a DB2) que ejecuta el paquete como valor que se va a utilizar

para la comprobación de autorización de sentencias de SQL dinámico y como valor inicial utilizado para la calificación implícita de referencias a objetos no calificados dentro de sentencias de SQL dinámico.

Comportamiento de vinculación
En el momento de la ejecución, DB2 utiliza todas las normas que se aplican al SQL estático para la autorización y calificación. Es decir, se toma el ID de autorización del propietario del paquete como valor que se va a utilizar para la comprobación de autorización de sentencias de SQL dinámico y el calificador por omisión del paquete para la calificación implícita de referencias a objetos no calificados dentro de sentencias de SQL dinámico.

Comportamiento de definición
El comportamiento de definición sólo se aplica si la sentencia de SQL dinámico está en un paquete que se ejecuta dentro del contexto de una rutina y el paquete se vinculó con DYNAMICRULES DEFINEBIND o DYNAMICRULES DEFINERUN. DB2 utiliza el ID de autorización del definidor de la rutina (no el vinculador de paquetes de la rutina) como valor que se va a utilizar para la comprobación de autorización de sentencias de SQL dinámico y para la calificación implícita de referencias a objetos no calificados dentro de sentencias de SQL dinámico contenidas en dicha rutina.

Comportamiento de invocación

El comportamiento de invocación sólo se aplica si la sentencia de SQL dinámico está en un paquete que se ejecuta dentro del contexto de una rutina y el paquete se vinculó con DYNAMICRULES INVOKEBIND o DYNAMICRULES INVOKERUN. DB2 utiliza el ID de autorización de sentencias actualmente en vigor cuando se invoca la rutina como valor que se va a utilizar para la comprobación de autorización de SQL dinámico y para la calificación implícita de referencias a objetos no calificados dentro de sentencias de SQL dinámico contenidas en dicha rutina. Esto se resume en la tabla siguiente:

Entorno de invocación	ID utilizado
Cualquier SQL estático	Valor implícito o explícito del propietario (OWNER) del paquete del que procede el SQL que invoca la rutina.
Utilizado en definición de vista o activador	Definidor de la vista o del activador.
SQL dinámico procedente de un paquete de comportamiento de ejecución	ID utilizado para efectuar la conexión inicial con DB2.
SQL dinámico procedente del paquete de comportamiento de definición	Definidor de la rutina que utiliza el paquete del que procede el SQL que invoca la rutina.
SQL dinámico procedente de un paquete de comportamiento de invocación	ID de autorización actual que invoca la rutina.

La tabla siguiente muestra la combinación del valor de DYNAMICRULES y el entorno de tiempo de ejecución que da lugar a cada comportamiento del SQL dinámico.

Tabla 18. 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 19. Definiciones de comportamientos de sentencias de SQL dinámico

Atributo de SQL dinámico	Valor correspondiente a atributos de SQL dinámico: comportamiento de vinculación	Valor correspondiente a atributos de SQL dinámico: comportamiento de ejecución	Valor correspondiente a atributos de SQL dinámico: comportamiento de definición	Valor correspondiente a atributos de SQL dinámico: comportamiento de invocación
ID de autorización	Valor implícito o explícito de la opción OWNER BIND	ID de usuario que ejecuta el paquete	Definidor de la rutina (no el propietario del paquete de la rutina)	ID de autorización de sentencias actual cuando se invoca la rutina.
Calificador por omisión para objetos no calificados	Valor implícito o explícito de la opción QUALIFIER BIND	Registro especial CURRENT SCHEMA	Definidor de la rutina (no el propietario del paquete de la rutina)	ID de autorización de sentencias actual cuando se invoca la rutina.
Puede ejecutar GRANT, REVOKE, ALTER, CREATE, DROP, COMMENT ON, RENAME, SET INTEGRITY y SET EVENT MONITOR STATE	No	Sí	No	No

Utilización de registros especiales para controlar el entorno de compilación de sentencias

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.

Cómo volver a crear paquetes utilizando el mandato BIND y un archivo de vinculación existente

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. Por omisión, el mandato PRECOMPILE crea un paquete. La vinculación se realiza de forma implícita en el momento de la precompilación, a no ser que se especifique la opción BINDFILE. La opción PACKAGE le permite especificar un nombre de paquete para el paquete que se crea en el momento de 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:

```
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)).

Revinculación de paquetes existentes con el mandato REBIND

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 o si las estadísticas de base de datos han cambiado desde la última vinculación. 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.

Consideraciones sobre la vinculación

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 Database para Linux, UNIX y Windows. 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 de la versión 8 se conecta a un servidor de la

versión 7.2, el cliente sólo podrá utilizar funciones de la versión 7.2. 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 Cliente DB2 de nivel inferior con el servidor de nivel superior y crear archivos de vinculación que se puedan suministrar y vincular al entorno de DB2 Database para Linux, UNIX y Windows 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.

Consideraciones acerca del bloqueo

Cuando desee desactivar el bloqueo de una aplicación de SQL incorporado y no esté disponible el código fuente, se debe volver a enlazar la aplicación con el mandato BIND y establecer la cláusula BLOCKING NO.

Las aplicaciones de SQL incorporado se deben volver a enlazar con el mandato BIND y estableciendo las cláusulas BLOCKING ALL o BLOCKING UNAMBIGUOUS para solicitar el bloqueo (si todavía no están enlazadas de este modo). Las aplicaciones de SQL incorporado recuperarán los valores LOB del servidor de fila en fila, después de que se hayan recuperado los bloques de fila del servidor.

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.

Mejoras en el rendimiento cuando se utiliza la opción REOPT del mandato BIND

la opción de vinculación REOPT puede mejorar significativamente el rendimiento de las aplicaciones de SQL incorporado. A continuación encontrará descripciones correspondientes a SQL estático e incorporado.

Efectos de REOPT en SQL estático

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

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

Efectos de REOPT en SQL dinámico

Cuando especifica la opción REOPT ALWAYS, DB2 aplaza la preparación de cualquier sentencia que contenga variables de lenguaje principal, marcadores de parámetros, variables globales y registros especiales hasta que DB2 encuentre una sentencia OPEN o EXECUTE; es decir, cuando los valores de esas variables pasan a ser conocidos. En ese momento, se genera el plan de acceso utilizando esos valores. Las peticiones OPEN o EXECUTE subsiguientes para la misma sentencia recompilarán la sentencia, reoptimizarán el plan de consulta utilizando el conjunto actual de valores de las variables y ejecutarán el plan de consulta recién creado.

La opción REOPT ONCE tiene un efecto similar, salvo que el plan se optimiza una sola vez utilizando los valores de las variables de lenguaje principal, marcadores de parámetros, variables globales y de registros especiales. Este plan se coloca en antememoria y es utilizado por peticiones subsiguientes.

Almacenamiento y mantenimiento de paquetes

Los paquetes se crean al precompilar/enlazar un programa de aplicación. El paquete contiene un plan de acceso optimizado que supervisa la ejecución de todas las sentencias de SQL que se encuentran dentro de la aplicación. Los tres tipos de privilegios relacionados con los paquetes son `CONTROL`, `EXECUTE` y `BIND` y sirven para filtrar el nivel de acceso aceptable. Se pueden crear varias versiones del mismo paquete especificando la opción `VERSION` en el momento de la compilación. Esta opción ayuda a evitar errores de falta de coincidencia de indicación de hora y permite ejecutar simultáneamente varias versiones de la aplicación.

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).

Resolución de nombres de tabla no calificados

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

- Cada usuario puede vincular su paquete con distintos parámetros de COLLECTION que utilizan distintos identificadores de autorización mediante los siguientes mandatos:

  ```
  CONNECT TO nombre_bd USER nombre_usuario
  BIND nombre_archivo COLLECTION nombre_esquema
  ```

  En el ejemplo anterior, *nombre_bd* es el nombre de la base de datos, *nombre_usuario* es el nombre del usuario y *nombre_archivo* es el nombre de la aplicación que se va a vincular. Tenga en cuenta que *nombre_usuario* y *nombre_esquema* 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. Si no se especifica COLLECTION, el calificador por omisión es el identificador de autorización que se utiliza al vincular el

paquete. Si se especifica COLLECTION, el *nombre_esquema* especificado es el calificador que se utilizará para objetos no calificados.

- 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.

Creación de aplicaciones de SQL incorporado mediante el script de creación de ejemplo

Los archivos utilizados para mostrar la creación de programas de ejemplo se conocen como archivos script en UNIX y Linux y como archivos de proceso por lotes en Windows. Nosotros los llamaremos genéricamente archivos de creación. Estos archivos contienen los mandatos de compilación y enlace recomendados para los compiladores de las plataformas soportadas.

DB2 proporciona archivos de creación para los lenguajes principales pertenecientes a las plataformas soportadas. Los archivos de creación están disponibles en el mismo directorio en el que se encuentran los ejemplos correspondientes a cada lenguaje. La tabla siguiente muestra los diferentes tipos de archivos de creación para crear diversos tipos de programas. A menos que se indique otra cosa, estos archivos de creación están pensados para los lenguajes soportados en todas las plataformas soportadas. Los archivos de creación tienen la extensión `.bat` (batch) en Windows, la cual no se muestra en la tabla. No existe extensión de archivo para las plataformas UNIX.

Tabla 20. Archivos de creación de DB2

Archivo de creación	**Tipos de programas creados**
bldapp	Programas de aplicación
bldrtn	Rutinas (procedimientos almacenados y funciones definidas por el usuario)
bldmc	Aplicaciones C/C++ de conexión múltiple
bldmt	Aplicaciones C/C++ de varias hebras
bldcli	Aplicaciones cliente de CLI para procedimientos SQL contenidos en el subdirectorio sqlproc de samples.

Nota: Por omisión, los scripts de ejemplo de bldapp para crear archivos ejecutables a partir de código fuente crearán archivos ejecutables de 64 bits.

La tabla siguiente lista los archivos de creación correspondientes a cada plataforma y lenguaje de programación, y los directorios donde están situados. En la documentación en línea, los nombres de los archivos de creación están enlazados dinámicamente con los archivos fuente de HTML. El usuario puede también acceder a los archivos de texto contenidos en los directorios samples correspondientes.

Tabla 21. Archivos de creación por lenguaje y plataforma

Plataforma —> Lenguaje	**AIX**	**HP-UX**	**Linux**	**Solaris**	**Windows**
C samples/c	bldapp bldrtn bldmt bldmc	bldapp bldrtn bldmt bldmc	bldapp bldrtn bldmt bldmc	bldapp bldrtn bldmt bldmc	bldapp.bat bldrtn.bat bldmt.bat bldmc.bat
C++ samples/cpp	bldapp bldrtn bldmt bldmc	bldapp bldrtn bldmt bldmc	bldapp bldrtn bldmt bldmc	bldapp bldrtn bldmt bldmc	bldapp.bat bldrtn.bat bldmt.bat bldmc.bat
IBM COBOL samples/cobol	bldapp bldrtn	n/a	n/a	n/a	bldapp.bat bldrtn.bat
Micro Focus COBOL samples/cobol_mf	bldapp bldrtn	bldapp bldrtn	bldapp bldrtn	bldapp bldrtn	bldapp.bat bldrtn.bat

Los archivos de creación se utilizan en la documentación sobre la creación de aplicaciones y rutinas porque muestran muy claramente las opciones de compilación y enlace que DB2 recomienda para los compiladores soportados Generalmente existen otras muchas opciones de compilación y enlace, que el usuario puede probar. Consulte la documentación del compilador para conocer todas las opciones de compilación y enlace proporcionadas. Además de crear los programas de ejemplo, el programador puede también crear sus propios programas mediante los archivos de creación. Los programas de ejemplo se pueden utilizar como plantillas que el usuario puede modificar como ayuda en el desarrollo de aplicaciones.

Una característica práctica de los archivos de creación es que están diseñados para crear un archivo fuente cuyo nombre puede ser cualquier nombre de archivo permitido por el compilador. En cambio, en los makefiles, los nombres de programa están codificados en el archivo. Los makefiles acceden a los archivos de creación para compilar y enlazar los programas que crean. Los archivos de creación utilizan la variable $1 en UNIX y Linux y la variable %1 en los sistemas operativos Windows para que internamente se utilice la variable en lugar del nombre de programa. Mediante un mayor número de estos nombres de variable se sustituyen otros argumentos que puedan ser necesarios.

Los archivos de creación permiten experimentar de forma rápida y sencilla, pues cada archivo está pensado para una clase específica de creación de programas, tales como la creación de programas autónomos, rutinas (procedimientos almacenados y funciones definidas por el usuario) o tipos de programa más especializados, tales como los programas de conexión múltiple o los programas de varias hebras. Se proporciona cada tipo de archivo de creación siempre que el compilador da soporte a la clase determinada de programa para el que está diseñado el archivo de creación.

El archivo objeto y el archivo ejecutable creados por un archivo de creación se sobrescriben cada vez que se crea un programa, aunque el archivo fuente no se modifique. No ocurre así cuando se utiliza un makefile. Por tanto, el programador puede volver a crear un programa existente sin tener que suprimir un archivo objeto o archivo ejecutable anterior, ni modificar el archivo fuente.

Los archivos de creación contienen un valor por omisión para la base de datos ″sample″ (base de datos de ejemplo). Si el usuario desea acceder a otra base de

datos, puede simplemente proporcionar otro parámetro para alterar temporalmente el valor por omisión. Si utilizan regularmente la otra base de datos, pueden codificar el nombre de esa base de datos, en lugar de `sample`, dentro del propio archivo de creación.

Para programas de SQL incorporado, salvo cuando se utilice el precompilador COBOL de IBM en Windows, los archivos de creación llamar a otro archivo, `embprep`, que contiene los pasos de precompilación y enlace para los programas de SQL incorporado. Para estos pasos pueden ser necesarios los parámetros opcionales correspondientes al ID de usuario y la contraseña, según el lugar donde se cree el programa de SQL incorporado.

Por último, el programador puede modificar los archivos de creación de acuerdo con sus necesidades. Además de cambiar el nombre de la base de datos dentro del archivo de creación (tal como se describió anteriormente), el programador puede codificar fácilmente otros parámetros dentro del archivo, cambiar opciones de compilación y enlace, o modificar la vía de instancia predefinida de DB2. El carácter simple, directo y específico del archivo de creación hace que el usuario pueda adaptarlo fácilmente a sus necesidades.

Programas de utilidad de comprobación de errores

El Cliente DB2 proporciona varios archivos de programas de utilidad. Estos archivos contienen funciones para la comprobación de errores y la impresión de información de errores. Los archivos de programa de utilidad correspondientes a cada lenguaje se encuentran en el directorio ″samples″. Cuando se utilizan junto con un programa de aplicación, los archivos de los programas de utilidad de comprobación de errores proporcionan información útil sobre errores y facilitan la depuración de los programas DB2. La mayoría de los programas de comprobación de errores utilizan las API de DB2 `GET SQLSTATE MESSAGE` (sqlogstt) y `GETERROR MESSAGE` (sqlaintp) para obtener la información pertinente de SQLSTATE y SQLCA referente a problemas encontrados al ejecutar el programa. El archivo de programa de utilidad de la CLI de DB2, `utilcli.c`, no hace uso de estas API de DB2; en su lugar utiliza sentencias equivalentes de la CLI de DB2. Todos los programas de utilidad de comprobación de errores imprimen mensajes de error descriptivos para ayudar al programador a comprender rápidamente el problema. Algunos programas DB2, tales como las rutinas (procedimientos almacenados y funciones definidas por el usuario), no necesitan hacer uso de los programas de utilidad.

A continuación se indican los archivos de los programas de utilidad de comprobación de errores utilizados por los compiladores soportados por DB2 para los diversos lenguajes de programación:

Tabla 22. Archivos del programa de utilidad de comprobación de errores por idioma

Lenguaje	Archivo fuente de SQL no incorporado	Archivo de cabecera de SQL no incorporado	Archivo fuente de SQL incorporado	Archivo de cabecera de SQL incorporado
C samples/c	`utilapi.c`	`utilapi.h`	`utilemb.sqc`	`utilemb.h`
C++ samples/cpp	`utilapi.C`	`utilapi.h`	`utilemb.sqC`	`utilemb.h`
IBM COBOL samples/cobol	`checkerr.cbl`	`n/a`	`n/a`	`n/a`
Micro Focus COBOL samples/cobol_mf	`checkerr.cbl`	`n/a`	`n/a`	`n/a`

Para poder utilizar las funciones del programa de utilidad, se debe primero compilar el archivo del programa de utilidad y luego enlazar su archivo objeto durante la creación del archivo ejecutable del programa destino. Tanto el `makefile` como los archivos de creación contenidos en los directorios `samples` realizan esas acciones para los programas que necesitan hacer uso de los programas de utilidad de comprobación de errores.

El ejemplo siguiente muestra cómo se utilizan los programas de utilidad de comprobación de errores en los programas DB2. El archivo de cabecera `utilemb.h` define la macro `EMB_SQL_CHECK`, para las funciones `SqlInfoPrint()` y `TransRollback()`:

```
/* macro para la comprobación del SQL incorporado */
#define EMB_SQL_CHECK( MSG_STR )           \
SqlInfoPrint(MSG_STR, &sqlca, __LINE__, __FILE__);     \
if (sqlca.sqlcode < 0)                                 \
{                                                      \
  TransRollback();                                     \
  return 1;                                            \
}
```

`SqlInfoPrint()` comprueba el SQLCODE y muestra la información disponible relacionada con el error específico encontrado. También indica el lugar donde se produjo el error en el código fuente. `TransRollback()` permite que el archivo de programa de utilidad retrotraiga sin peligro una transacción donde se ha producido un error. Utiliza la sentencia de SQL incorporado `EXEC SQL ROLLBACK`. El ejemplo siguiente nuestra cómo el programa de C `dbuse` invoca las funciones del programa de utilidad utilizando la macro, y proporciona el valor `"Delete with host variables -- Execute"` para el parámetro `MSG_STR` de la función `SqlInfoPrint()`:

```
EXEC SQL DELETE FROM org
  WHERE deptnumb = :hostVar1 AND
        division = :hostVar2;
EMB_SQL_CHECK("Delete with host variables -- Execute");
```

La macro `EMB_SQL_CHECK` hace que si la sentencia `DELETE` falla, la transacción se retrotraiga sin peligro, y se imprima el mensaje de error apropiado.

El programador puede utilizar estos programas de utilidad de comprobación de errores y utilizarlos como modelo al crear sus propios programas DB2.

Creación de aplicaciones y rutinas escritas en C y C++

Se proporcionan scripts de creación para distintas plataformas de sistemas operativos con el producto para facilitar la creación de aplicaciones de SQL incorporado en C y C++. Además de los scripts de creación utilizados para crear aplicaciones, se proporciona un script específico, `bldrtn`, que sirve para crear rutinas (procedimientos almacenados y funciones definidas por el usuario). Para aplicaciones y rutinas escritas en VisualAge, se utilizan archivos de configuración para crear las aplicaciones. Los ejemplos de aplicaciones C suministrados varían desde ejemplos de nivel de guía de aprendizaje o de cliente hasta ejemplo de nivel de instancia y se encuentran en el directorio `sqllib/samples/c` para UNIX y en el directorio `sqllib\samples\c` para Windows.

Opciones de compilar y enlazar para C y C++

Opciones de compilación y enlace de aplicaciones de API de DB2 y de SQL incorporado de C de AIX:

Las siguientes son las opciones de compilación y enlace que recomienda DB2 para crear aplicaciones de DB2 y C de SQL incorporado con el compilador C de AIX de IBM, tal como muestra el script de compilación `bldapp`.

Opciones de compilación y enlace para bldapp

Opciones de compilación: **xlc** El compilador IBM XL C/C++. **$EXTRA_CFLAG** Contiene "-q64" para una instancia en que está habilitado el soporte de 64 bits; de lo contrario, no contiene ningún valor. **-I$DB2PATH/include** Especifica la ubicación de los archivos de inclusión de DB2. Por ejemplo: `$HOME/sqllib/include`. **-c** Hace que se efectúe solamente la compilación, no la edición de enlaces. La compilación y la edición de enlaces son pasos separados.
Opciones de enlace: **xlc** Hace que el compilador se utilice como interfaz del editor de enlaces. **$EXTRA_CFLAG** Contiene "-q64" para una instancia en que está habilitado el soporte de 64 bits; de lo contrario, no contiene ningún valor. **-o $1** Especifica el programa ejecutable. **$1.o** Especifica el archivo objeto del programa. **utilemb.o** Si es un programa de SQL incorporado, incluir el archivo objeto del programa de utilidad de SQL incorporado para la comprobación de errores. **utilapi.o** Si crea un programa que no es de SQL incorporado, incluir el archivo objeto del programa de utilidad de la API de DB2 para la comprobación de errores. **-ldb2** Enlazar con la biblioteca de DB2. **-L$DB2PATH/$LIB** Especifica la ubicación de las bibliotecas compartidas de tiempo de ejecución de DB2. Por ejemplo: `$HOME/sqllib/$LIB`. Si el usuario no especifica la opción `-L`, el compilador utiliza esta vía de acceso: `/usr/lib:/lib`. Consulte la documentación del compilador para conocer otras opciones de compilador.

Opciones de compilación y enlace de aplicaciones de la API de administración de DB2 y de SQL incorporado en C++ de AIX:

Las siguientes son las opciones de compilación y enlace que recomienda DB2 para crear aplicaciones de SQL incorporado de C++ y de la API administrativa de DB2 con el compilador XL C/C++ de AIX de IBM, como se muestra en el script de compilación `bldapp`.

Opciones de compilación y enlace para bldapp

Opciones de compilación: **xlC** El compilador IBM XL C/C++. **EXTRA_CFLAG** Contiene "-q64" para una instancia en que está habilitado el soporte de 64 bits; de lo contrario, no contiene ningún valor. **-I$DB2PATH/include** Especifica la ubicación de los archivos de inclusión de DB2. Por ejemplo: `$HOME/sqllib/include`. **-c** Hace que se efectúe solamente la compilación, no la edición de enlaces. La compilación y la edición de enlaces son pasos separados.
Opciones de enlace: **xlC** Hace que el compilador se utilice como interfaz del editor de enlaces. **EXTRA_CFLAG** Contiene "-q64" para una instancia en que está habilitado el soporte de 64 bits; de lo contrario, no contiene ningún valor. **-o $1** Especifica el programa ejecutable. **$1.o** Especifica el archivo objeto del programa. **utilapi.o** Hace que se incluya el archivo objeto del programa de utilidad de la API para programas que no contienen SQL incorporado. **utilemb.o** Incluye el archivo objeto del programa de utilidad de SQL incorporado para los programas de SQL incorporado. **-ldb2** Enlazar con la biblioteca de DB2. **-L$DB2PATH/$LIB** Especifica la ubicación de las bibliotecas compartidas de tiempo de ejecución de DB2. Por ejemplo: `$HOME/sqllib/$LIB`. Si el usuario no especifica la opción `-L`, el compilador utiliza esta vía de acceso `/usr/lib:/lib`. Consulte la documentación del compilador para conocer otras opciones de compilador.

Opciones de compilación y enlace para aplicaciones C de HP-UX:

La tabla siguiente muestra las opciones de compilación y enlace que DB2 recomienda para crear aplicaciones C de SQL incorporado y de la API de DB2 con el compilador C de HP-UX, tal como muestra el script de creación `bldapp`.

Opciones de compilación y enlace para bldapp

Opciones de compilación:

cc Indica el compilador C.

$EXTRA_CFLAG
Si la plataforma HP-UX es IA64 y está habilitado el soporte de 64 bits, este distintivo contiene el valor **+DD64**; si está habilitado el soporte de 32 bits, contiene el valor **+DD32**. Si la plataforma HP-UX es PA-RISC y está habilitado el soporte de 64 bits, contiene el valor **+DA2.0W**. Para el soporte de 32 bits en una plataforma PA-RISC, este distintivo contiene el valor **+DA2.0N**.

+DD64 Se debe utilizar para generar código de 64 bits para HP-UX en IA64.

+DD32 Se debe utilizar para generar código de 32 bits para HP-UX en IA64.

+DA2.0W
Se debe utilizar para generar código de 64 bits para HP-UX en PA-RISC.

+DA2.0N
Se debe utilizar para generar código de 32 bits para HP-UX en PA-RISC.

-Ae Habilita la modalidad ampliada ANSI de HP.

-I$DB2PATH/include
Especifica la ubicación de los archivos de inclusión de DB2.

-c Hace que se efectúe solamente la compilación, no la edición de enlaces. La compilación y la edición de enlaces son pasos separados.

Opciones de enlace:

cc Hace que el compilador se utilice como interfaz del editor de enlaces.

$EXTRA_CFLAG
Si la plataforma HP-UX es IA64 y está habilitado el soporte de 64 bits, este distintivo contiene el valor **+DD64**; si está habilitado el soporte de 32 bits, contiene el valor **+DD32**. Si la plataforma HP-UX es PA-RISC y está habilitado el soporte de 64 bits, contiene el valor **+DA2.0W**. Para el soporte de 32 bits en una plataforma PA-RISC, este distintivo contiene el valor **+DA2.0N**.

+DD64 Se debe utilizar para generar código de 64 bits para HP-UX en IA64.

+DD32 Se debe utilizar para generar código de 32 bits para HP-UX en IA64.

+DA2.0W
Se debe utilizar para generar código de 64 bits para HP-UX en PA-RISC.

+DA2.0N
Se debe utilizar para generar código de 32 bits para HP-UX en PA-RISC.

-o $1 Especifica el ejecutable.

$1.o Especifica el archivo objeto del programa.

utilemb.o
Si es un programa de SQL incorporado, incluir el archivo objeto del programa de utilidad de SQL incorporado para la comprobación de errores.

utilapi.o
Si es un programa de SQL no incorporado, incluir el archivo objeto del programa de utilidad de la API de DB2 para la comprobación de errores.

$EXTRA_LFLAG
Especificar la vía de acceso de tiempo de ejecución. Si se establece, para 32 bits contiene el valor `-Wl,+b$HOME/sqllib/lib32`, y para 64 bits contiene `-Wl,+b$HOME/sqllib/lib64`. Si no se establece, no contiene ningún valor.

-L$DB2PATH/$LIB
Especifica la ubicación de las bibliotecas compartidas de tiempo de ejecución de DB2. Para 32 bits: `$HOME/sqllib/lib32`; para 64 bits: `$HOME/sqllib/lib64`.

-ldb2 Enlazar con la biblioteca de DB2.

Consulte la documentación del compilador para conocer otras opciones de compilador.

Opciones de compilación y enlace para aplicaciones C++ de HP-UX:

La tabla siguiente muestra las opciones de compilación y enlace que DB2 recomienda para crear aplicaciones en C++ de SQL incorporado y de la API de DB2 con el compilador C++ de HP-UX, tal como muestra el script de creación `bldapp`.

Opciones de compilación y enlace para bldapp

Opciones de compilación:

aCC Indica el compilador aC++ de HP.

$EXTRA_CFLAG
Si la plataforma HP-UX es IA64 y está habilitado el soporte de 64 bits, este distintivo contiene el valor **+DD64**; si está habilitado el soporte de 32 bits, contiene el valor **+DD32**. Si la plataforma HP-UX es PA-RISC y está habilitado el soporte de 64 bits, contiene el valor **+DA2.0W**. Para el soporte de 32 bits en una plataforma PA-RISC, este distintivo contiene el valor **+DA2.0N**.

+DD64 Se debe utilizar para generar código de 64 bits para HP-UX en IA64.

+DD32 Se debe utilizar para generar código de 32 bits para HP-UX en IA64.

+DA2.0W
Se debe utilizar para generar código de 64 bits para HP-UX en PA-RISC.

+DA2.0N
Se debe utilizar para generar código de 32 bits para HP-UX en PA-RISC.

-ext Permite el uso de diversas extensiones de C++, que incluyen el soporte para "long long".

-I$DB2PATH/include
Especifica la ubicación de los archivos de inclusión de DB2. Por ejemplo: `$HOME/sqllib/include`

-c Hace que se efectúe solamente la compilación, no la edición de enlaces. La compilación y la edición de enlaces son pasos separados.

Opciones de enlace:

aCC Hace que el compilador aC++ de HP se utilice como interfaz del editor de enlaces.

$EXTRA_CFLAG
Si la plataforma HP-UX es IA64 y está habilitado el soporte de 64 bits, este distintivo contiene el valor **+DD64**; si está habilitado el soporte de 32 bits, contiene el valor **+DD32**. Si la plataforma HP-UX es PA-RISC y está habilitado el soporte de 64 bits, contiene el valor **+DA2.0W**. Para el soporte de 32 bits en una plataforma PA-RISC, este distintivo contiene el valor **+DA2.0N**.

+DD64 Se debe utilizar para generar código de 64 bits para HP-UX en IA64.

+DD32 Se debe utilizar para generar código de 32 bits para HP-UX en IA64.

+DA2.0W
Se debe utilizar para generar código de 64 bits para HP-UX en PA-RISC.

+DA2.0N
Se debe utilizar para generar código de 64 bits para HP-UX en PA-RISC.

-o $1 Especifica el ejecutable.

$1.o Especifica el archivo objeto del programa.

utilemb.o
Si es un programa de SQL incorporado, incluir el archivo objeto del programa de utilidad de SQL incorporado para la comprobación de errores.

utilapi.o
Si es un programa de SQL no incorporado, incluir el archivo objeto del programa de utilidad de la API de DB2 para la comprobación de errores.

$EXTRA_LFLAG
Especificar la vía de acceso de tiempo de ejecución. Si se establece, para 32 bits contiene el valor "`-Wl,+b$HOME/sqllib/lib32`", y para 64 bits: "`-Wl,+b$HOME/sqllib/lib64`". Si no se establece, no contiene ningún valor.

-L$DB2PATH/$LIB
Especifica la ubicación de las bibliotecas compartidas de tiempo de ejecución de DB2. Para 32 bits: `$HOME/sqllib/lib32`; para 64 bits: `$HOME/sqllib/lib64`.

-ldb2 Enlazar con la biblioteca de DB2.

Consulte la documentación del compilador para conocer otras opciones de compilador.

Opciones de compilación y enlace para aplicaciones C de Linux:

La tabla siguiente muestra las opciones de compilación y enlace que DB2 recomienda para crear aplicaciones C de SQL incorporado y de la API de DB2 con el compilador C de Linux, tal como muestra el script de creación `bldapp`.

Opciones de compilación y enlace para bldapp

Opciones de compilación:

$CC El compilador gcc o xlc_r.

$EXTRA_C_FLAGS
Contiene uno de los siguientes valores:

- -m31 en Linux sólo para zSeries, para crear una biblioteca de 32 bits;
- -m32 en Linux para x86, x86_64 y POWER, para crear una biblioteca de 32 bits;
- -m64 en Linux para zSeries, POWER, x86_64, para crear una biblioteca de 64 bits; o
- Ningún valor en Linux para IA64, para crear una biblioteca de 64 bits.

-I$DB2PATH/include
Especifica la ubicación de los archivos de inclusión de DB2.

-c Hace que se efectúe solamente la compilación, no la edición de enlaces. Este archivo de script tiene pasos separados para la compilación y la edición de enlaces.

Opciones de enlace:

$CC El compilador gcc o xlc_r; hace que el compilador se utilice como interfaz del editor de enlaces.

$EXTRA_C_FLAGS
Contiene uno de los siguientes valores:

- -m31 en Linux sólo para zSeries, para crear una biblioteca de 32 bits;
- -m32 en Linux para x86, x86_64 y POWER, para crear una biblioteca de 32 bits;
- -m64 en Linux para zSeries, POWER, x86_64, para crear una biblioteca de 64 bits; o
- Ningún valor en Linux para IA64, para crear una biblioteca de 64 bits.

-o $1 Especifica el ejecutable.

$1.o Especifica el archivo objeto.

utilemb.o
Si es un programa de SQL incorporado, incluir el archivo objeto del programa de utilidad de SQL incorporado para la comprobación de errores.

utilapi.o
Si es un programa de SQL no incorporado, incluir el archivo objeto del programa de utilidad de la API de DB2 para la comprobación de errores.

$EXTRA_LFLAG
Para 32 bits contiene el valor ″-Wl,-rpath,$DB2PATH/lib32″, y para 64 bits contiene el valor ″-Wl,-rpath,$DB2PATH/lib64″.

-L$DB2PATH/$LIB
Especifica la ubicación de las bibliotecas estáticas y compartidas de DB2 durante el enlace. Por ejemplo, para 32 bits: $HOME/sqllib/lib32, y para 64 bits: $HOME/sqllib/lib64.

-ldb2 Enlazar con la biblioteca de DB2.

Consulte la documentación del compilador para conocer otras opciones de compilador.

Opciones de compilación y enlace para aplicaciones C++ de Linux:

La tabla siguiente muestra las opciones de compilación y enlace que DB2 recomienda para crear aplicaciones en C++ de SQL incorporado y de la API de DB2 con el compilador C++ de Linux, tal como muestra el script de creación `bldapp`.

Opciones de compilación y enlace para bldapp

Opciones de compilación: **g++** Indica el compilador C++ de GNU/Linux. **$EXTRA_C_FLAGS** Contiene uno de los siguientes valores: • `-m31` en Linux sólo para zSeries, para crear una biblioteca de 32 bits; • `-m32` en Linux para x86, x86_64 y POWER, para crear una biblioteca de 32 bits; • `-m64` en Linux para zSeries, POWER, x86_64, para crear una biblioteca de 64 bits; o • Ningún valor en Linux para IA64, para crear una biblioteca de 64 bits. **-I$DB2PATH/include** Especifica la ubicación de los archivos de inclusión de DB2. **-c** Hace que se efectúe solamente la compilación, no la edición de enlaces. Este archivo de script tiene pasos separados para la compilación y la edición de enlaces.
Opciones de enlace: **g++** Hace que el compilador se utilice como interfaz del editor de enlaces. **$EXTRA_C_FLAGS** Contiene uno de los siguientes valores: • `-m31` en Linux sólo para zSeries, para crear una biblioteca de 32 bits; • `-m32` en Linux para x86, x86_64 y POWER, para crear una biblioteca de 32 bits; • `-m64` en Linux para zSeries, POWER, x86_64, para crear una biblioteca de 32 bits; o • Ningún valor en Linux para IA64, para crear una biblioteca de 64 bits. **-o $1** Especifica el ejecutable. **$1.o** Incluir el archivo objeto del programa. **utilemb.o** Si es un programa de SQL incorporado, incluir el archivo objeto del programa de utilidad de SQL incorporado para la comprobación de errores. **utilapi.o** Si es un programa de SQL no incorporado, incluir el archivo objeto del programa de utilidad de la API de DB2 para la comprobación de errores. **$EXTRA_LFLAG** Para 32 bits contiene el valor ″-Wl,-rpath,$DB2PATH/lib32″, y para 64 bits contiene el valor ″-Wl,-rpath,$DB2PATH/lib64″. **-L$DB2PATH/$LIB** Especifica la ubicación de las bibliotecas estáticas y compartidas de DB2 durante el enlace. Por ejemplo, para 32 bits: `$HOME/sqllib/lib32`, y para 64 bits: `$HOME/sqllib/lib64`. **-ldb2** Enlazar con la biblioteca de DB2. Consulte la documentación del compilador para conocer otras opciones de compilador.

Opciones de compilación y enlace para aplicaciones C de Solaris:

La tabla siguiente muestra las opciones de compilación y enlace que DB2 recomienda para crear aplicaciones C de SQL incorporado y de la API de DB2 con el compilador C de Forte, tal como muestra el script de creación `bldapp`.

Opciones de compilación y enlace para bldapp

Opciones de compilación: **cc** Indica el compilador C. **-xarch=$CFLAG_ARCH** Esta opción asegura que el compilador creará ejecutables válidos cuando establezca un enlace con `libdb2.so`. El valor de $CFLAG_ARCH es ″v8plusa″ para la modalidad de 32 bits, o ″v9″ para la modalidad de 64 bits. **-I$DB2PATH/include** Especifica la ubicación de los archivos de inclusión de DB2. Por ejemplo: `$HOME/sqllib/include` **-c** Hace que se efectúe solamente la compilación, no la edición de enlaces. Este script tiene pasos separados para la compilación y la edición de enlaces.
Opciones de enlace: **cc** Hace que el compilador se utilice como interfaz del editor de enlaces. **-xarch=$CFLAG_ARCH** Esta opción asegura que el compilador creará ejecutables válidos cuando establezca un enlace con `libdb2.so`. El valor de $CFLAG_ARCH es ″v8plusa″ para la modalidad de 32 bits, o ″v9″ para la modalidad de 64 bits. **-mt** Crea enlaces cuando se utiliza el soporte de varias hebras. Esta opción es necesaria para enlazar con libdb2. **Nota:** Si se utilizan hebras de POSIX, las aplicaciones DB2 se deben también enlazar con `-lpthread`, tengan o no varias hebras. **-o $1** Especifica el ejecutable. **$1.o** Incluir el archivo objeto del programa. **utilemb.o** Si es un programa de SQL incorporado, incluir el archivo objeto del programa de utilidad de SQL incorporado para la comprobación de errores. **utilapi.o** Si crea un programa que no es de SQL incorporado, incluir el archivo objeto del programa de utilidad de la API de DB2 para la comprobación de errores. **-L$DB2PATH/$LIB** Especifica la ubicación de las bibliotecas estáticas y compartidas de DB2 durante el enlace. Por ejemplo, para 32 bits: `$HOME/sqllib/lib32`, y para 64 bits: `$HOME/sqllib/lib64`. **$EXTRA_LFLAG** Especifica la ubicación de las bibliotecas compartidas de DB2 durante la ejecución. Para 32 bits contiene el valor ″-R$DB2PATH/lib32″, y para 64 bits contiene el valor ″-R$DB2PATH/lib64″. **-ldb2** Enlazar con la biblioteca de DB2. Consulte la documentación del compilador para conocer otras opciones de compilador.

Opciones de compilación y enlace para aplicaciones C++ de Solaris:

La tabla siguiente muestra las opciones de compilación y enlace que DB2 recomienda para crear aplicaciones en C++ de SQL incorporado y de la API de DB2 con el compilador C++ de Forte, tal como muestra el script de creación `bldapp`.

Opciones de compilación y enlace para bldapp

Opciones de compilación: **CC** Indica el compilador C++. **-xarch=$CFLAG_ARCH** Esta opción asegura que el compilador creará ejecutables válidos cuando establezca un enlace con `libdb2.so`. El valor de $CFLAG_ARCH es ″v8plusa″ para la modalidad de 32 bits, o ″v9″ para la modalidad de 64 bits. **-I$DB2PATH/include** Especifica la ubicación de los archivos de inclusión de DB2. Por ejemplo: `$HOME/sqllib/include` **-c** Hace que se efectúe solamente la compilación, no la edición de enlaces. Este script tiene pasos separados para la compilación y la edición de enlaces.
Opciones de enlace: **CC** Hace que el compilador se utilice como interfaz del editor de enlaces. **-xarch=$CFLAG_ARCH** Esta opción asegura que el compilador creará ejecutables válidos cuando establezca un enlace con `libdb2.so`. El valor de $CFLAG_ARCH es ″v8plusa″ para la modalidad de 32 bits, o ″v9″ para la modalidad de 64 bits. **-mt** Crea enlaces cuando se utiliza el soporte de varias hebras. Esta opción es necesaria para enlazar con libdb2. **Nota:** Si se utilizan hebras de POSIX, las aplicaciones DB2 se deben también enlazar con `-lpthread`, tengan o no varias hebras. **-o $1** Especifica el ejecutable. **$1.o** Incluir el archivo objeto del programa. **utilemb.o** Si es un programa de SQL incorporado, incluir el archivo objeto del programa de utilidad de SQL incorporado para la comprobación de errores. **utilapi.o** Si es un programa de SQL no incorporado, incluir el archivo objeto del programa de utilidad de la API de DB2 para la comprobación de errores. **-L$DB2PATH/$LIB** Especifica la ubicación de las bibliotecas estáticas y compartidas de DB2 durante el enlace. Por ejemplo, para 32 bits: `$HOME/sqllib/lib32`, y para 64 bits: `$HOME/sqllib/lib64`. **$EXTRA_LFLAG** Especifica la ubicación de las bibliotecas compartidas de DB2 durante la ejecución. Para 32 bits contiene el valor ″-R$DB2PATH/lib32″, y para 64 bits contiene el valor ″-R$DB2PATH/lib64″. **-ldb2** Enlazar con la biblioteca de DB2. Consulte la documentación del compilador para conocer otras opciones de compilador.

Opciones de compilación y enlace para aplicaciones C y C++ de Windows:

A continuación se muestran las opciones de compilación y enlace recomendadas para DB2 para crear aplicaciones de SQL incorporado y de la API de DB2 en C y C++ en Windows con el compilador Microsoft Visual C++, tal como se muestra en el archivo de proceso por lotes `bldapp.bat`.

Opciones de compilación y enlace para bldapp

Opciones de compilación: **%BLDCOMP%** Variable del compilador. El valor por omisión es `cl`, que indica el compilador Microsoft Visual C++. Puede tener también el valor `icl`, el compilador Intel C++ para para aplicaciones de 32 ó 64 bits, o `ecl`, el compilador Intel C++ para aplicaciones Itanium de 64 bits. **-Zi** Habilitar la información de depuración. **-Od** Inhabilitar las optimizaciones. Es más fácil utilizar un depurador si la optimización está inhabilitada. **-c** Hace que se efectúe solamente la compilación, no la edición de enlaces. El archivo de proceso por lotes tiene pasos separados para la compilación y la edición de enlaces. **-W2** Emitir mensajes de aviso, de error, de error grave y de error no recuperable. **-DWIN32** Opción de compilador necesaria para los sistemas operativos Windows.
Opciones de enlace: **link** Indica la utilización del enlazador para la edición de enlaces. **-debug** Hace que se incluya información de depuración. **-out:%1.exe** Especifique un nombre de archivo. **%1.obj** Incluir el archivo objeto. **utilemb.obj** Si es un programa de SQL incorporado, incluir el archivo objeto del programa de utilidad de SQL incorporado para la comprobación de errores. **utilapi.obj** Si crea un programa que no es de SQL incorporado, incluir el archivo objeto del programa de utilidad de la API de DB2 para la comprobación de errores. **db2api.lib** Enlazar con la biblioteca DB2. Consulte la documentación del compilador para conocer otras opciones de compilador.

Creación de aplicaciones en C o C++ mediante el script de creación de ejemplo (UNIX)

DB2 proporciona scripts de creación para compilar y enlazar programas de SQL incorporado y de la API de administración de DB2 en C o C++. Se encuentran en el directorio `sqllib/samples/c` para aplicaciones en C y en el directorio `sqllib/samples/cpp` para aplicaciones en C++, junto con programas de ejemplo que se pueden crear con estos archivos.

El archivo de creación, `bldapp` contiene los mandatos para crear un programa de aplicación DB2.

El primer parámetro, $1, especifica el nombre del archivo fuente. Éste es el único parámetro necesario para los programas de la API de administración de DB2 que no contienen SQL incorporado. Para crear programas de SQL incorporado es necesaria una conexión con la base de datos, por lo que también se proporcionan otros tres parámetros opcionales: el segundo parámetro, $2, especifica el nombre de la base de datos con la que se desea conectar; el tercer parámetro, $3, especifica el ID de usuario correspondiente a la base de datos, y $4 especifica la contraseña.

Para un programa de SQL incorporado, `bldapp` pasa los parámetros al script de precompilación y vinculación, `embprep`. Si no se proporciona un nombre de base de datos, se utiliza la base de datos por omisión `sample`. Los parámetros de ID de usuario y contraseña sólo son necesarios si la instancia donde se crea el programa es diferente de la instancia donde reside la base de datos.

Los ejemplos siguientes muestran cómo crear y ejecutar programas de la API de administración de DB2 y aplicaciones de SQL incorporado.

Creación y ejecución de aplicaciones de la API de administración de DB2

Para crear el programa de ejemplo de la API de administración de DB2, `cli_info`, a partir del archivo fuente `cli_info.c` para C y `cli_info.C` para ++, especifique:

```
bldapp cli_info
```

El resultado es un archivo ejecutable, `cli_info`.

Para ejecutar el archivo ejecutable, escriba el nombre del ejecutable:

```
cli_info
```

Creación y ejecución de aplicaciones de SQL incorporado

- Hay tres formas de crear la aplicación de SQL incorporado, `tbmod` , a partir del archivo fuente `tbmod.sqc` para C y `tbmod.sqC` para C++,:
  1. Si conecta con la base de datos "sample" en la misma instancia, especifique:
     ```
     bldapp tbmod
     ```
  2. Si conecta con otra base de datos en la misma instancia, especifique también el nombre de la base de datos:
     ```
     bldapp tbmod basedatos
     ```
  3. Si conecta con una base de datos de otra instancia, especifique también el ID de usuario y la contraseña de la instancia de base de datos:
     ```
     bldapp tbmod basedatos IDusuario contraseña
     ```
     El resultado es un archivo ejecutable, `tbmod`.
- Existen tres maneras de ejecutar esta aplicación de SQL incorporado:
  1. Si desea acceder a la base de datos `sample` en la misma instancia, especifique simplemente el nombre del ejecutable:
     ```
     tbmod
     ```
  2. Si desea acceder a otra base de datos en la misma instancia, especifique el nombre del ejecutable y el nombre de la base de datos:
     ```
     tbmod basedatos
     ```
  3. Si desea acceder a una base de datos de otra instancia, especifique el nombre del ejecutable, el nombre de la base de datos, y el ID de usuario y la contraseña de la instancia de base de datos:
     ```
     tbmod basedatos IDusuario contraseña
     ```

Creación de aplicaciones C/C++ en Windows

DB2 proporciona scripts de creación para compilar y enlazar programas de la API de DB2 y de SQL incorporado en C/C++. Estos archivos residen en los directorios `sqllib/samples/c` y `sqllib\samples\cpp`, junto con programas de ejemplo que se pueden crear a partir de esos archivos.

El archivo de proceso por lotes `bldapp.bat` contiene los mandatos para crear programas de la API de DB2 y de SQL incorporado. Utiliza un máximo de cuatro parámetros como entrada, que están representados dentro del archivo de proceso por lotes por las variables `%1`, `%2`, `%3` y `%4`.

El primer parámetro, `%1`, especifica el nombre del archivo fuente. Éste es el único parámetro necesario para los programas que no contienen SQL incorporado. Para crear programas de SQL incorporado es necesaria una conexión con la base de datos, por lo que también se proporcionan otros tres parámetros: el segundo parámetro, `%2`, especifica el nombre de la base de datos con la que se desea conectar; el tercer parámetro, `%3`, especifica el ID de usuario correspondiente a la base de datos, y `%4` especifica la contraseña.

Para un programa de SQL incorporado, `bldapp` pasa los parámetros al archivo de precompilación y vinculación, `embprep.bat`. Si no se proporciona un nombre de base de datos, se utiliza la base de datos por omisión `sample`. Los parámetros de ID de usuario y contraseña sólo son necesarios si la instancia donde se crea el programa es diferente de la instancia donde reside la base de datos.

- **Creación y ejecución de aplicaciones de SQL incorporado**

  Existen tres maneras para crear la aplicación de SQL incorporado `tbmod`, a partir del archivo fuente C `tbmod.sqc` de `sqllib\samples\c`, o partir del archivo C++ `tbmod.sqx` de `sqllib\samples\cpp`:

  – Si conecta con la base de datos "sample" en la misma instancia, especifique:

    ```
    bldapp tbmod
    ```

  – Si conecta con otra base de datos en la misma instancia, especifique también el nombre de la base de datos:

    ```
    bldapp tbmod basedatos
    ```

  – Si conecta con una base de datos de otra instancia, especifique también el ID de usuario y la contraseña de la instancia de base de datos:

    ```
    bldapp tbmod basedatos IDusuario contraseña
    ```

  El resultado es un archivo ejecutable, `tbmod.exe`.

  Existen tres maneras de ejecutar esta aplicación de SQL incorporado:

  – Si desea acceder a la base de datos `sample` en la misma instancia, especifique simplemente el nombre del ejecutable:

    ```
    tbmod
    ```

  – Si desea acceder a otra base de datos en la misma instancia, especifique el nombre del ejecutable y el nombre de la base de datos:

    ```
    tbmod basedatos
    ```

  – Si desea acceder a una base de datos de otra instancia, especifique el nombre del ejecutable, el nombre de la base de datos, y el ID de usuario y la contraseña de la instancia de base de datos:

    ```
    tbmod basedatos IDusuario contraseña
    ```

- **Creación y ejecución de aplicaciones de varias hebras**

  Las aplicaciones C/C++ de varias hebras en Windows se tienen que compilar con las opciones `-MT` o `-MD`. La opción `-MT` se enlazará utilizando la biblioteca estática `LIBCMT.LIB`, y `-MD` se enlazará utilizando la biblioteca dinámica

MSVCRT.LIB. El binario enlazado con -MD será más pequeño pero dependiente de MSVCRT.DLL, mientras que el binario enlazado con -MT será más grande pero independiente respecto al momento de ejecución.

El archivo de proceso por lotes bldmt.bat utiliza la opción -MT para crear un programa de varias hebras. Todas las otras opciones de compilación y enlace son iguales a las utilizadas por el archivo de proceso por lotes bldapp.bat para crear aplicaciones autónomas normales.

Para crear el programa de varias hebras de ejemplo, dbthrds, a partir de los archivos fuente samples\c\dbthrds.sqc o samples\cpp\dbthrds.sqx, especifique:

```
bldmt dbthrds
```

El resultado es un archivo ejecutable, dbthrds.exe.

Existen tres maneras de ejecutar esta aplicación de varias hebras:

- Si accede a la base de datos sample en la misma instancia, especifique simplemente el nombre del ejecutable (sin la extensión):

  ```
  dbthrds
  ```

- Si desea acceder a otra base de datos en la misma instancia, especifique el nombre del ejecutable y el nombre de la base de datos:

  ```
  dbthrds basedatos
  ```

- Si desea acceder a una base de datos de otra instancia, especifique el nombre del ejecutable, el nombre de la base de datos, y el ID de usuario y la contraseña de la instancia de base de datos:

  ```
  dbthrds basedatos IDusuario contraseña
  ```

Los ejemplos siguientes muestran cómo crear y ejecutar programas de la API de DB2 y aplicaciones de SQL incorporado.

Para crear el programa de ejemplo de la API de DB2 de SQL no incorporado, cli_info, a partir del archivo fuente cli_info.c, situado en sqllib\samples\c, o a partir del archivo fuente cli_info.cxx, situado en sqllib\samples\cpp, especifique:

```
bldapp cli_info
```

El resultado es un archivo ejecutable, cli_info.exe. Puede ejecutar el archivo ejecutable escribiendo el nombre del ejecutable (sin la extensión) en la línea de mandatos:

```
cli_info
```

Creación de aplicaciones de SQL incorporado escritas en VisualAge C++ con archivos de configuración

VisualAge C++ tiene un compilador incremental y un compilador de proceso por lotes. Mientras que el compilador de proceso por lotes utiliza archivos de creación y de compilación, el compilador incremental utiliza archivos de configuración en su lugar. Consulte la documentación de VisualAge C++ Versión 5.0 para conocer más sobre este tema.

DB2 proporciona archivos de configuración para los diferentes tipos de programas DB2 que el usuario puede crear con el compilador VisualAge C++.

Para utilizar un archivo de configuración de DB2, primero debe asignar una variable de entorno al nombre del programa que desea compilar. A continuación, compile el programa con un mandato proporcionado por VisualAge C++. Estos son los temas que describen cómo utilizar los archivos de configuración proporcionados por DB2 para compilar los diversos tipos de programas:

- Creación de aplicaciones de SQL incorporado y de la API administrativa de DB2 en C o C++ con archivos de configuración (AIX)
- "Creación de procedimientos almacenados de SQL incorporado en C o C++ con archivos de configuración" en *Desarrollo de rutinas definidas por el usuario (SQL y externas)*
- "Creación de funciones definidas por el usuario en C o C++ con archivos de configuración (AIX)" en *Desarrollo de rutinas definidas por el usuario (SQL y externas)*

Creación de aplicaciones de SQL incorporado y de API de DB2 en C o C++ con archivos de configuración (AIX)

Los archivos de configuración son básicamente scripts de creación utilizados para crear aplicaciones escritas en VisualAge. El script hace varias cosas, que incluyen comprobación de errores, conexión a una base de datos, precompilación del código, establecimiento de la vía de acceso en la que se puede acceder a DB2 y finalmente generación del archivo ejecutable.

Los ejemplos siguientes muestran cómo crear y ejecutar programas de la API de administración de DB2 y aplicaciones de SQL incorporado con archivos de configuración.

- **Creación de aplicaciones de SQL incorporado C y C++ con archivos de configuración**

  El archivo de configuración `emb.icc`, contenido en `sqllib/samples/c` y `sqllib/samples/cpp`, le permite crear aplicaciones de SQL incorporado para DB2 utilizando C o C++ en AIX.Para utilizar el archivo de configuración a fin de crear la aplicación de SQL incorporado `tbmod`, a partir del archivo fuente `tbmod.sqc`, siga estos pasos:

  1. Asigne a la variable de entorno EMB el nombre del programa, de este modo:
     - En el shell Bash o Korn:
       ```
       export EMB=tbmod
       ```
     - En el shell C:
       ```
       setenv EMB tbmod
       ```
  2. Si su directorio de trabajo contiene un archivo `emb.ics`, resultante de la creación de un programa diferente con el archivo `emb.icc`, suprima el archivo `emb.ics` mediante este mandato:
     ```
     rm emb.ics
     ```
     Si para el mismo programa que ahora creará de nuevo ya existe un archivo `emb.ics`, no es necesario suprimir el archivo.
  3. Compile el programa de ejemplo, especificando:
     ```
     vacbld emb.icc
     ```

     Nota: El mandato `vacbld` está incluido en VisualAge C++.

     El resultado es un archivo ejecutable, `tbmod`. Puede ejecutar el programa entrando el nombre del ejecutable:
     ```
     tbmod
     ```

- **Creación de aplicaciones de API de DB2 C y C++ con archivos de configuración**

  El archivo de configuración, `api.icc` en `sqllib/samples/c` y en `sqllib/samples/cpp`, le permite crear programas de la API de administración de DB2 en C o C++ en AIX.

Para utilizar el archivo de configuración para crear el programa de ejemplo de la API de administración de DB2, `cli_info`, a partir del archivo fuente `cli_info.c`, haga lo siguiente:

1. Asigne a la variable de entorno API el nombre del programa, de este modo:
   - En el shell Bash o Korn:
     ```
     export API=cli_info
     ```
   - En el shell C:
     ```
     setenv API cli_info
     ```
2. Si su directorio de trabajo contiene un archivo `api.ics`, resultante de la creación de un programa diferente con el archivo `api.icc`, suprima el archivo `api.ics` mediante este mandato:
   ```
   rm api.ics
   ```

   Si para el mismo programa que ahora creará de nuevo ya existe un archivo `api.ics`, no es necesario suprimir el archivo.
3. Compile el programa de ejemplo, especificando:
   ```
   vacbld api.icc
   ```

   Nota: El mandato `vacbld` está incluido en VisualAge C++.

   El resultado es un archivo ejecutable, `cli_info`. Puede ejecutar el programa entrando el nombre del ejecutable:
   ```
   cli_info
   ```

Creación de aplicaciones C/C++ de conexión múltiple en Windows

DB2 proporciona scripts de creación para compilar y enlazar programas C y C++ de SQL incorporado y de los programas de API de DB2. Estos archivos residen en los directorios `sqllib/samples/c` y `sqllib\samples\cpp`, junto con programas de ejemplo que se pueden crear a partir de esos archivos.

El archivo de proceso por lotes, `bldmc.bat`, contiene los mandatos para crear un programa de conexión múltiple de DB2, que requiere dos bases de datos. Las opciones de compilación y enlace son las mismas que las utilizadas en el archivo `bldapp.bat`.

El primer parámetro, `%1`, especifica el nombre del archivo fuente. El segundo parámetro, `%2`, especifica el nombre de la primera base de datos a la que desea conectarse. El tercer parámetro, `%3`, especifica la segunda base de datos a la que desea conectarse. Todos estos parámetros son obligatorios.

Nota: El script de creación codifica los valores por omisión ″sample″ y ″sample2″ como nombres de la base de datos (`%2` y `%3`, respectivamente) por lo que, si está utilizando el script de creación y acepta estos valores, sólo tendrá que especificar el nombre del programa (parámetro `%1`). Si utiliza el script `bldmc.bat` debe especificar los tres parámetros.

No se requieren parámetros opcionales para una conexión local, pero sí para conectar con un servidor desde un cliente remoto. Estos parámetros son: `%4` y `%5` para especificar, respectivamente, el ID de usuario y la contraseña para la primera base de datos; y `%6` y `%7` para especificar, respectivamente, el ID de usuario y la contraseña para la segunda base de datos.

Para el programa de ejemplo de conexión múltiple, dbmcon.exe, se requieren dos bases de datos. Si aún no se ha creado la base de datos sample, la puede crear entrando db2sampl en la línea de mandatos de una ventana de mandatos de DB2. La segunda base de datos, aquí llamada sample2, se puede crear mediante uno de los mandatos siguientes:

Si crea la base de datos localmente:

```
db2 create db sample2
```

Si crea la base de datos remotamente:

```
db2 attach to <nombre_nodo>
db2 create db sample2
db2 detach
db2 catalog db sample2 as sample2 at node <nombre_nodo>
```

donde<nombre_nodo> es el nodo en que reside la base de datos.

Una conexión múltiple también requiere que se esté ejecutando el receptor TCP/IP. Para asegurarse que es así, haga lo siguiente:

1. Establezca la variable de entorno DB2COMM en TCP/IP de la forma siguiente:

   ```
   db2set DB2COMM=TCPIP
   ```

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

   ```
   db2 update dbm cfg using SVCENAME <nombre del servicio TCP/IP>
   ```

   Cada instancia tiene un nombre de servicio TCP/IP listado en el archivo de servicios. Si no puede localizarlo o no tiene el permiso de archivo para cambiar el archivo de servicios, pida ayuda al administrador.

3. Detenga y reinicie el gestor de bases de datos para que estos cambios entren en vigor:

   ```
   db2stop
   db2start
   ```

El programa dbmcon.exe se crea a partir de cinco archivos de los directorios samples\c o samples\cpp:

dbmcon.sqc o dbmcon.sqx
: Archivo fuente principal para conectar con ambas bases de datos.

dbmcon1.sqc o dbmcon1.sqx
: Archivo fuente para crear un paquete vinculado a la primera base de datos.

dbmcon1.h
: Archivo de cabecera para dbmcon1.sqc o dbmcon1.sqx, incluido en el archivo fuente principal, dbmcon.sqc o dbmcon.sqx, para acceder a las sentencias de SQL para crear y eliminar una tabla que se debe vincular a la primera base de datos.

dbmcon2.sqc o dbmcon2.sqx
: Archivo fuente para crear un paquete vinculado a la segunda base de datos.

dbmcon2.h
: Archivo de cabecera para dbmcon2.sqc o dbmcon2.sqx, incluido en el

archivo fuente principal, `dbmcon.sqc` o `dbmcon.sqx`, para acceder a las sentencias de SQL para crear y eliminar una tabla que se debe vincular a la segunda base de datos.

Para crear el programa de ejemplo de conexión múltiple, `dbmcon.exe`, se entre lo siguiente:

```
bldmc dbmcon sample sample2
```

El resultado es un archivo ejecutable, `dbmcon.exe`.

Para ejecutar el archivo ejecutable, entre el nombre del ejecutable sin la extensión:

```
dbmcon
```

El programa muestra una confirmación en una fase para dos bases de datos.

Creación de aplicaciones y rutinas escritas en COBOL

Se proporcionan scripts de creación para distintas plataformas de sistemas operativos con el producto para facilitar la creación de aplicaciones de SQL incorporado en COBOL. Además de los scripts de creación utilizados para crear aplicaciones, se proporciona un script específico, `bldrtn`, que sirve para crear rutinas (procedimientos almacenados y funciones definidas por el usuario). Cuando trabaje con aplicaciones escrita en el lenguaje Micro Focus COBOL en Linux, asegúrese de configurar el compilador de modo que pueda acceder a ciertas bibliotecas compartidas de COBOL. Se suministran ejemplos IBM COBOL que se encuentran en el directorio `sqllib/samples/cobol` para UNIX y en el directorio `sqllib\samples\cobol` para Windows; para los directorios de ejemplos de Micro Focus COBOL, sustituya 'cobol' al final de la vía de acceso por 'cobol_mf'.

Opciones de compilar y enlazar para COBOL

Opciones de compilación y enlace para aplicaciones IBM COBOL de AIX:

A continuación se muestran las opciones de compilación y enlace recomendadas para DB2 para crear aplicaciones de SQL incorporado y de la API de DB2 en COBOL con el compilador IBM COBOL para AIX, tal como se muestra en el script de creación `bldapp`.

Opciones de compilación y enlace para bldapp

Opciones de compilación:	
cob2	El compilador IBM COBOL para AIX.
-qpgmname\(mixed\)	Indica al compilador que permita llamadas a los puntos de entrada de biblioteca utilizando nombres con mayúsculas y minúsculas mezcladas.
-qlib	Indica al compilador que procese las sentencias COPY.
-I$DB2PATH/include/cobol_a	Especifica la ubicación de los archivos de inclusión de DB2. Por ejemplo: `$HOME/sqllib/include/cobol_a`.
-c	Hace que se efectúe solamente la compilación, no la edición de enlaces. La compilación y la edición de enlaces son pasos separados.

Opciones de enlace:

cob2 Hace que el compilador se utilice como interfaz del editor de enlaces.

-o $1 Especifica el programa ejecutable.

$1.o Especifica el archivo objeto del programa.

checkerr.o
Hace que se incluya el archivo objeto de programa de utilidad para la comprobación de errores.

-L$DB2PATH/$LIB
Especifica la ubicación de las bibliotecas compartidas de tiempo de ejecución de DB2. Por ejemplo: `$HOME/sqllib/lib32`.

-ldb2 Enlazar con la biblioteca del gestor de bases de datos.

Consulte la documentación del compilador para conocer otras opciones de compilador.

Opciones de compilación y enlace para aplicaciones Micro Focus COBOL de AIX:

A continuación se muestran las opciones de compilación y enlace que se recomiendan para DB2 para crear aplicaciones de SQL incorporado y de la API de DB2 en COBOL con el compilador Micro Focus COBOL en AIX, tal como se muestra en el script de creación `bldapp`. Tenga en cuenta que los archivos de inclusión de DB2 MicroFocus COBOL se localizan configurando la variable de entorno COBCPY, de modo que no se necesita el distintivo -I en el paso de compilación. Consulte el script bldapp para ver un ejemplo.

Opciones de compilación y enlace para bldapp

Opciones de compilación:

cob El compilador MicroFocus COBOL.

-c Hace que se efectúe solamente la compilación, no la edición de enlaces.

$EXTRA_COBOL_FLAG="-C MFSYNC"
Habilita el soporte de 64 bits.

-x Cuando se utiliza con `-c`, crea un archivo objeto.

Opciones de enlace:

cob Hace que el compilador se utilice como interfaz del editor de enlaces.

-x Crea un programa ejecutable.

-o $1 Especifica el programa ejecutable.

$1.o Especifica el archivo objeto del programa.

-L$DB2PATH/$LIB
Especifica la ubicación de las bibliotecas compartidas de tiempo de ejecución de DB2. Por ejemplo: `$HOME/sqllib/lib32`.

-ldb2 Enlazar con la biblioteca de DB2.

-ldb2gmf
Establece un enlace con la biblioteca del gestor de excepciones de DB2 correspondiente a Micros Focus COBOL.

Consulte la documentación del compilador para conocer otras opciones de compilador.

Opciones de compilación y enlace para aplicaciones Micro Focus COBOL de HP-UX:

A continuación se muestran las opciones de compilación y enlace que se recomiendan para DB2 para crear aplicaciones de SQL incorporado y de la API de DB2 en COBOL con el compilador Micro Focus COBOL en HP-UX, tal como se muestra en el script de creación `bldapp`.

Opciones de compilación y enlace para bldapp

Opciones de compilación:
cob Indica el compilador Micro Focus COBOL.
-cx Compila y crea el módulo objeto.
$EXTRA_COBOL_FLAG Contiene ″-C MFSYNC″ si la plataforma HP-UX es IA64 y el soporte de 64 bits está habilitado.
Opciones de enlace:
cob Hace que el compilador se utilice como interfaz del editor de enlaces.
-x Especifica un programa ejecutable.
$1.o Incluir el archivo objeto del programa.
checkerr.o Incluir el archivo objeto del programa de utilidad para la comprobación de errores.
-L$DB2PATH/$LIB Especifica la ubicación de las bibliotecas compartidas de tiempo de ejecución de DB2.
-ldb2 Enlazar con la biblioteca de DB2.
-ldb2gmf Establece un enlace con la biblioteca del gestor de excepciones de DB2 correspondiente a Micros Focus COBOL.
Consulte la documentación del compilador para conocer otras opciones de compilador.

Opciones de compilación y enlace para aplicaciones Micro Focus COBOL de Solaris:

A continuación se muestran las opciones de compilación y enlace que se recomiendan para DB2 para crear aplicaciones de SQL incorporado y de la API de DB2 en COBOL con el compilador Micro Focus COBOL en Solaris, tal como se muestra en el script de creación `bldapp`.

Opciones de compilación y enlace para bldapp

Opciones de compilación:
cob Indica el compilador Micro Focus COBOL.
$EXTRA_COBOL_FLAG Para el soporte de 64 bits, contiene el valor ″-C MFSYNC″; de lo contrario, no contiene ningún valor.
-cx Compila y crea el módulo objeto.

Opciones de enlace:

cob Hace que el compilador se utilice como interfaz del editor de enlaces.

-x Especifica un programa ejecutable.

$1.o Incluir el archivo objeto del programa.

checkerr.o
Hace que se incluya el archivo objeto de programa de utilidad para la comprobación de errores.

-L$DB2PATH/$LIB
Especifica la ubicación de las bibliotecas estáticas y compartidas de DB2 durante el enlace. Por ejemplo: `$HOME/sqllib/lib64`.

-ldb2 Enlazar con la biblioteca de DB2.

-ldb2gmf
Establece un enlace con la biblioteca del gestor de excepciones de DB2 correspondiente a Micros Focus COBOL.

Consulte la documentación del compilador para conocer otras opciones de compilador.

Opciones de compilación y enlace para aplicaciones Micro Focus COBOL de Linux:

A continuación se muestran las opciones de compilación y enlace que se recomiendan para DB2 para crear aplicaciones de SQL incorporado y de la API de DB2 en COBOL con el compilador Micro Focus COBOL en Linux, tal como se muestra en el script de creación `bldapp`.

Opciones de compilación y enlace para bldapp

Opciones de compilación:

cob Indica el compilador Micro Focus COBOL.

-cx Compila y crea el módulo objeto.

$EXTRA_COBOL_FLAG
Para el soporte de 64 bits, contiene el valor "-C MFSYNC"; de lo contrario, no contiene ningún valor.

Opciones de enlace:

cob Hace que el compilador se utilice como interfaz del editor de enlaces.

-x Especifica un programa ejecutable.

-o $1 Incluir el ejecutable.

$1.o Incluir el archivo objeto del programa.

checkerr.o
Incluir el archivo objeto del programa de utilidad para la comprobación de errores.

-L$DB2PATH/$LIB
Especifica la ubicación de las bibliotecas compartidas de tiempo de ejecución de DB2.

-ldb2 Enlazar con la biblioteca de DB2.

-ldb2gmf
Establece un enlace con la biblioteca del gestor de excepciones de DB2 correspondiente a Micros Focus COBOL.

Consulte la documentación del compilador para conocer otras opciones de compilador.

Opciones de compilación y enlace para aplicaciones IBM COBOL de Windows:

A continuación se muestran las opciones de compilación y enlace recomendadas para DB2 para crear aplicaciones de SQL incorporado y de la API de DB2 en COBOL en Windows con el compilador IBM VisualAge COBOL, tal como se muestra en el archivo de proceso por lotes `bldapp.bat`.

Opciones de compilación y enlace para bldapp

Opciones de compilación:

cob2 El compilador IBM VisualAge COBOL.

-qpgmname(mixed)
Indica al compilador que permita llamadas a los puntos de entrada de biblioteca utilizando nombres con mayúsculas y minúsculas mezcladas.

-c Hace que se efectúe solamente la compilación, no la edición de enlaces. La compilación y la edición de enlaces son pasos separados.

-qlib Indica al compilador que procese las sentencias COPY.

-I*path* Especifica la ubicación de los archivos de inclusión de DB2. Por ejemplo: `-I"%DB2PATH%\include\cobol_a"`.

%EXTRA_COMPFLAG%
Si "set `IBMCOB_PRECOMP=true`" no tiene comentarios, el precompilador IBM COBOL se utilizará para precompilar el SQL incorporado. Se invocará con una de las siguientes formulaciones, en función de los parámetros de entrada:

-q"SQL('database sample CALL_RESOLUTION DEFERRED')"
precompilar utilizando la base de datos de ejemplo por omisión, y diferir la resolución de la llamada.

-q"SQL('database %2 CALL_RESOLUTION DEFERRED')"
precompilar utilizando una base de datos especificada por el usuario, y diferir la resolución de la llamada.

-q"SQL('database %2 user %3 using %4 CALL_RESOLUTION DEFERRED')"
precompilar utilizando una base de datos, un ID de usuario y una contraseña especificados por el usuario, y diferir la resolución de la llamada. Éste es el formato para el acceso de clientes remotos.

Opciones de enlace:

cob2 Utilizar el compilador como interfaz del editor de enlaces.

%1.obj Incluir el archivo objeto del programa.

checkerr.obj
Incluir el archivo objeto del programa de utilidad de comprobación de errores.

db2api.lib
Enlazar con la biblioteca de DB2.

Consulte la documentación del compilador para conocer otras opciones de compilador.

Opciones de compilación y enlace para aplicaciones Micro Focus COBOL de Windows:

A continuación se muestran las opciones de compilación y enlace recomendadas para DB2 para crear aplicaciones de SQL incorporado y de la API de DB2 en COBOL en Windows con el compilador Micro Focus COBOL, tal como se muestra en el archivo de proceso por lotes `bldapp.bat`.

Opciones de compilación y enlace para bldapp

Opción de compilación:

cobol Indica el compilador Micro Focus COBOL.

Opciones de enlace:

cbllink
Utilizar el enlazador para la edición de enlaces.

-l Enlazar con la biblioteca lcobol.

checkerr.obj
Enlazar con el archivo objeto del programa de utilidad de comprobación de errores.

db2api.lib
Enlazar con la biblioteca DB2 de la API.

Consulte la documentación del compilador para conocer otras opciones de compilador.

Configuraciones del compilador COBOL

Configuración del compilador IBM COBOL en AIX:

Debe seguir los pasos siguientes si desarrolla aplicaciones que contienen SQL incorporado y llamadas a la API de DB2, y está utilizando el compilador IBM COBOL Set para AIX.

- Cuando precompile la aplicación utilizando el mandato de procesador de línea de mandatos `db2 prep`, utilice la opción `target ibmcob`.
- No utilice caracteres de tabulación en los archivos fuente.
- Puede utilizar las palabras clave `PROCESS` y `CBL` en la primera línea de los archivos fuente para definir opciones de compilación.
- Si la aplicación sólo contiene SQL incorporado y ninguna llamada a la API de DB2, no es necesario que utilice la opción de compilación `pgmname(mixed)`. Si especifica llamadas a la API de DB2, debe utilizar la opción de compilación `pgmname(mixed)`.
- Si utiliza la opción "System/390 host data type support" del compilador IBM COBOL Set para AIX, los archivos de inclusión de DB2 correspondientes a las aplicaciones están contenidos en este directorio:

  ```
  $HOME/sqllib/include/cobol_i
  ```

  Si crea programas DB2 de ejemplo utilizando los archivos de script proporcionados, la vía de acceso de los archivos de inclusión especificada en los archivos de script se debe cambiar para que apunte al directorio `cobol_i` y no al directorio `cobol_a`.

  Si NO utiliza la opción "System/390 host data type support" del compilador IBM COBOL Set para AIX, o está utilizando una versión anterior de este compilador, los archivos de inclusión de DB2 correspondientes a las aplicaciones están contenidos en este directorio:

  ```
  $HOME/sqllib/include/cobol_a
  ```

  Especifique nombres de archivo de COPY de modo que incluyan la extensión `.cbl`, de la manera siguiente:

  ```
  COPY "sql.cbl".
  ```

Configuración del compilador IBM COBOL en Windows:

Si desarrolla aplicaciones que contienen SQL incorporado y llamadas a la API de DB2 y está utilizando el compilador IBM VisualAge COBOL, hay varias consideraciones a tener en cuenta.

- Cuando precompile la aplicación utilizando el precompilador de DB2 y use el mandato de procesador de línea de mandatos `db2 prep`, utilice la opción `target ibmcob`.
- No utilice caracteres de tabulación en los archivos fuente.
- Utilice las palabras clave `PROCESS` y `CBL` en los archivos fuente para definir opciones de compilación. Coloque las palabras clave en las columnas 8 a la 72 solamente.
- Si la aplicación sólo contiene SQL incorporado y ninguna llamada a la API de DB2, no es necesario que utilice la opción de compilación `pgmname(mixed)`. Si especifica llamadas a la API de DB2, debe utilizar la opción de compilación `pgmname(mixed)`.
- Si utiliza la opción ″System/390 host data type support″ del compilador IBM VisualAge COBOL, los archivos de inclusión de DB2 correspondientes a las aplicaciones se encuentran en el siguiente directorio:

  ```
  %DB2PATH%\include\cobol_i
  ```

  Si crea programas DB2 de ejemplo utilizando los archivos de proceso por lotes proporcionados, la vía de acceso de los archivos de inclusión especificada en los archivos de proceso por lotes se debe cambiar para que apunte al directorio `cobol_i` y no al directorio `cobol_a`.

  Si NO utiliza la opción ″System/390 host data type support″ del compilador IBM VisualAge COBOL, o si utiliza una versión anterior de este compilador, los archivos de inclusión de DB2 correspondientes a las aplicaciones se encuentran en el siguiente directorio:

  ```
  %DB2PATH%\include\cobol_a
  ```

  El directorio `cobol_a` es el directorio por omisión.
- Especifique nombres de archivo de COPY de modo que incluyan la extensión `.cbl`, de la manera siguiente:

  ```
  COPY "sql.cbl".
  ```

Configuración del compilador Micro Focus COBOL en Windows:

Si desarrolla aplicaciones que contienen SQL incorporado y llamadas a la API de DB2, y está utilizando el compilador Micro Focus, existen varias consideraciones para tener en cuenta.

- Cuando precompile la aplicación utilizando el mandato `db2 prep` del procesador de línea de mandatos, utilice la opción `target mfcob`.
- Asegúrese de que la variable de entorno LIB apunta a `%DB2PATH%\lib` utilizando el siguiente mandato:

  ```
  set LIB="%DB2PATH%\lib;%LIB%"
  ```
- Los archivos de COPY para DB2 correspondientes a Micro Focus COBOL residen en `%DB2PATH%\include\cobol_mf`. Defina la variable de entorno `COBCPY` para que incluya ese directorio del siguiente modo:

  ```
  set COBCPY="%DB2PATH%\include\cobol_mf;%COBCPY%"
  ```

  Debe asegurarse de que las variables de entorno antes mencionadas estén establecidas permanentemente en los valores del `Sistema`. Esto se puede comprobar siguiendo estos pasos:

  1. Abra el **Panel de control**
  2. Seleccione **Sistema**
  3. Seleccione la pestaña **Avanzado**
  4. Pulse en **Variables de entorno**

5. Compruebe en la lista **Variables del sistema** si aparecen las variables de entorno necesarias. Si no están, añádalas a la lista **Variables del sistema**

Establecerlas en los valores de `Usuario`, en un indicador de mandatos o en un script no es suficiente.

Las llamadas a las API de DB2 se deben realizar utilizando el convenio de llamada 74. El precompilador DB2 COBOL inserta automáticamente una cláusula CALL-CONVENTION en un párrafo SPECIAL-NAMES. Si el párrafo SPECIAL-NAMES no existe, el precompilador DB2 COBOL lo crea, de esta manera:

```
Identification Division
Program-ID. "static".
special-names.
    call-convention 74 is DB2API.
```

Además, el precompilador inserta automáticamente el símbolo DB2API, que sirve para identificar el convenio de llamada, a continuación de la palabra clave ″call″ cada vez que se llama a una API de DB2. Esto ocurre, por ejemplo, cada vez que el precompilador emite una llamada a una API de DB2 a partir de una sentencia de SQL incorporado.

Si las llamadas a las API de DB2 se realizan en una aplicación que no está precompilada, es necesario crear manualmente un párrafo SPECIAL-NAMES en la aplicación, similar al proporcionado más arriba. Si llama a una API de DB2 directamente, necesitará añadir manualmente el símbolo DB2API a continuación de la palabra clave ″call″.

Configuración del compilador Micro Focus COBOL en Linux:

Para ejecutar rutinas de Micro Focus COBOL, el editor de enlaces de Linux debe poder acceder a determinadas bibliotecas compartidas de Java, y DB2 debe poder cargar estas bibliotecas. Debido a que el programa que realiza esta carga se ejecuta con privilegios setuid, sólo buscará las bibliotecas dependientes en `/usr/lib`.

Cree enlaces simbólicos con `/usr/lib` para las bibliotecas compartidas de COBOL. Esto se debe hacer como usuario root. La manera más sencilla de hacerlo es enlazando todos los archivos de bibliotecas de COBOL de `$COBDIR/lib` con `/usr/lib`:

```
ln -s $COBDIR/lib/libcob* /usr/lib
```

donde `$COBDIR` es el lugar en que está instalado Micro Focus COBOL, normalmente `/opt/lib/mfcobol`.

Éstos son los mandatos para enlazar cada uno de los archivos individuales (suponiendo que Micro Focus COBOL está instalado en `/opt/lib/mfcobol`):

```
ln -s /opt/lib/mfcobol/lib/libcobrts.so /usr/lib
ln -s /opt/lib/mfcobol/lib/libcobrts_t.so /usr/lib
ln -s /opt/lib/mfcobol/lib/libcobrts.so.2 /usr/lib
ln -s /opt/lib/mfcobol/lib/libcobrts_t.so.2 /usr/lib
ln -s /opt/lib/mfcobol/lib/libcobcrtn.so /usr/lib
ln -s /opt/lib/mfcobol/lib/libcobcrtn.so.2 /usr/lib
ln -s /opt/lib/mfcobol/lib/libcobmisc.so /usr/lib
ln -s /opt/lib/mfcobol/lib/libcobmisc_t.so /usr/lib
ln -s /opt/lib/mfcobol/lib/libcobmisc.so.2 /usr/lib
ln -s /opt/lib/mfcobol/lib/libcobmisc_t.so.2 /usr/lib
ln -s /opt/lib/mfcobol/lib/libcobscreen.so /usr/lib
ln -s /opt/lib/mfcobol/lib/libcobscreen.so.2 /usr/lib
ln -s /opt/lib/mfcobol/lib/libcobtrace.so /usr/lib
```

```
ln -s /opt/lib/mfcobol/lib/libcobtrace_t.so /usr/lib
ln -s /opt/lib/mfcobol/lib/libcobtrace.so.2 /usr/lib
ln -s /opt/lib/mfcobol/lib/libcobtrace_t.so.2 /usr/lib
```

Es necesario hacer lo siguiente en cada instancia de DB2.

- Cuando precompile la aplicación utilizando el mandato de procesador de línea de mandatos `db2 prep`, utilice la opción `target mfcob`.
- Debe incluir el directorio del archivo COPY de DB2 para COBOL en la variable de entorno COBCOPY de Micro Focus COBOL. La variable de entorno COBCPY especifica la ubicación de los archivos COPY. Los archivos COPY de DB2 correspondientes a Micro Focus COBOL residen en `sqllib/include/cobol_mf`, dentro del directorio de la instancia de la base de datos.

  Para incluir el directorio en la variable de entorno, especifique:

  – En el shell bash o Korn:

  ```
  export COBCPY=$HOME/sqllib/include/cobol_mf:$COBDIR/cpylib
  ```

  – En el shell C:

  ```
  setenv COBCPY $HOME/sqllib/include/cobol_mf:$COBDIR/cpylib
  ```

- Actualizar la variable de entorno:

  – En el shell bash o Korn:

  ```
  export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$HOME/sqllib/lib:$COBDIR/lib
  ```

  – En el shell C:

  ```
  setenv LD_LIBRARY_PATH $LD_LIBRARY_PATH:$HOME/sqllib/lib:$COBDIR/lib
  ```

- Establecer la Lista de entornos de DB2:

  ```
  db2set DB2ENVLIST="COBDIR LD_LIBRARY_PATH"
  ```

Nota: Es posible que desee establecer COBCPY, COBDIR y LD_LIBRARY_PATH en `.bashrc`, `.kshrc` (según el shell que se utilice), `.bash_profile`, `.profile` (según el shell que se utilice) o en `.login`. .

Configuración del compilador Micro Focus COBOL en AIX:

Siga los pasos siguientes si desarrolla aplicaciones que contienen SQL incorporado y llamadas a la API de DB2 y utiliza el compilador Micro Focus COBOL.

- Cuando precompile la aplicación utilizando el mandato de procesador de línea de mandatos `db2 prep`, utilice la opción `target mfcob`.
- Debe incluir el directorio del archivo COPY de DB2 para COBOL en la variable de entorno COBCOPY de Micro Focus COBOL. La variable de entorno COBCPY especifica la ubicación de los archivos COPY. Los archivos COPY de DB2 correspondientes a Micro Focus COBOL residen en `sqllib/include/cobol_mf`, dentro del directorio de la instancia de la base de datos.

  Para incluir el directorio en la variable de entorno, especifique:

  – En el shell bash o Korn:

  ```
  export COBCPY=$COBCPY:$HOME/sqllib/include/cobol_mf
  ```

  – En el shell C:

  ```
  setenv COBCPY $COBCPY:$HOME/sqllib/include/cobol_mf
  ```

  Nota: Puede definir COBCPY dentro del archivo `.profile` o `.login`.

Configuración del compilador Micro Focus COBOL en HP-UX:

Si desarrolla aplicaciones que contienen SQL incorporado y llamadas a la API de DB2, y está utilizando el compilador Micro Focus COBOL, existen varias consideraciones para tener en cuenta.

- Cuando precompile la aplicación utilizando el mandato de procesador de línea de mandatos `db2 prep`, utilice la opción `target mfcob`.
- Debe incluir el directorio del archivo COPY de DB2 para COBOL en la variable de entorno COBCOPY de Micro Focus COBOL. La variable de entorno COBCPY especifica la ubicación de los archivos COPY. Los archivos COPY de DB2 correspondientes a Micro Focus COBOL residen en `sqllib/include/cobol_mf`, dentro del directorio de la instancia de la base de datos.

  Para incluir el directorio en la variable de entorno,

  – en el shell bash o Korn, especifique:

  ```
  export COBCPY=$COBCPY:$HOME/sqllib/include/cobol_mf
  ```

  – en el shell C, especifique:

  ```
  setenv COBCPY ${COBCPY}:${HOME}/sqllib/include/cobol_mf
  ```

  Nota: Puede definir COBCPY dentro del archivo `.profile` o `.login`.

Configuración del compilador Micro Focus COBOL en Solaris:

Si desarrolla aplicaciones que contienen SQL incorporado y llamadas a la API de DB2, y está utilizando el compilador Micro Focus COBOL, debe tener en cuenta varias consideraciones.

- Cuando precompile la aplicación utilizando el mandato de procesador de línea de mandatos `db2 prep`, utilice la opción `target mfcob`.
- Debe incluir el directorio del archivo COPY de DB2 para COBOL en la variable de entorno COBCOPY de Micro Focus COBOL. La variable de entorno COBCPY especifica la ubicación de los archivos COPY. Los archivos COPY de DB2 correspondientes a Micro Focus COBOL residen en `sqllib/include/cobol_mf`, dentro del directorio de la instancia de la base de datos.

  Para incluir el directorio en la variable de entorno, especifique:

  – En el shell bash o Korn:

  ```
  export COBCPY=$COBCPY:$HOME/sqllib/include/cobol_mf
  ```

  – En el shell C:

  ```
  setenv COBCPY $COBCPY:$HOME/sqllib/include/cobol_mf
  ```

  Nota: Puede definir COBCPY dentro del archivo `.profile`.

Creación de aplicaciones IBM COBOL en AIX

DB2 proporciona scripts de creación para compilar y enlazar programas IBM COBOL de SQL incorporado y programas de la API de administración de DB2. Estos scripts residen en el directorio `sqllib/samples/cobol`, junto con programas de ejemplo que se pueden crear a partir de esos archivos.

El archivo de creación, `bldapp`, contiene los mandatos para crear un programa de aplicación DB2.

El primer parámetro, `$1`, especifica el nombre del archivo fuente. Éste es el único parámetro necesario para los programas que no contienen SQL incorporado. Para crear programas de SQL incorporado es necesaria una conexión con la base de datos, por lo que también se proporcionan otros tres parámetros opcionales: el segundo parámetro, `$2`, especifica el nombre de la base de datos con la que se

desea conectar; el tercer parámetro, $3, especifica el ID de usuario correspondiente a la base de datos, y $4 especifica la contraseña.

Para un programa de SQL incorporado, bldapp pasa los parámetros al script de precompilación y vinculación, embprep. Si no se proporciona un nombre de base de datos, se utiliza la base de datos por omisión sample. Los parámetros de ID de usuario y contraseña sólo son necesarios si la instancia donde se crea el programa es diferente de la instancia donde reside la base de datos.

Para crear el programa de ejemplo de SQL no incorporado, client, a partir del archivo fuente client.cbl, especifique:

```
bldapp client
```

El resultado es un archivo ejecutable, client. Para ejecutar el archivo ejecutable sobre la base de datos sample, especifique:

```
client
```

Creación y ejecución de aplicaciones de SQL incorporado

- A partir del archivo fuente updat.sqb, puede crear la aplicación de SQL incorporado, updat, de tres maneras:
  1. Si conecta con la base de datos "sample" en la misma instancia, especifique:
     ```
     bldapp updat
     ```
  2. Si conecta con otra base de datos en la misma instancia, especifique también el nombre de la base de datos:
     ```
     bldapp updat basedatos
     ```
  3. Si conecta con una base de datos de otra instancia, especifique también el ID de usuario y la contraseña de la instancia de base de datos:
     ```
     bldapp updat basedatos IDusuario contraseña
     ```
     El resultado es un archivo ejecutable, updat.
- Existen tres maneras de ejecutar esta aplicación de SQL incorporado:
  1. Si desea acceder a la base de datos sample en la misma instancia, especifique simplemente el nombre del ejecutable:
     ```
     updat
     ```
  2. Si desea acceder a otra base de datos en la misma instancia, especifique el nombre del ejecutable y el nombre de la base de datos:
     ```
     updat basedatos
     ```
  3. Si desea acceder a una base de datos de otra instancia, especifique el nombre del ejecutable, el nombre de la base de datos, y el ID de usuario y la contraseña de la instancia de base de datos:
     ```
     bldapp updat basedatos IDusuario contraseña
     ```

Creación de aplicaciones Micro Focus COBOL de UNIX

DB2 proporciona scripts de creación para compilar y enlazar programas Micro Focus COBOL de SQL incorporado y de la API de DB2. Estos scripts residen en el directorio sqllib/samples/cobol_mf, junto con programas de ejemplo que se pueden crear a partir de esos archivos.

El archivo de creación, bldapp, contiene los mandatos para crear un programa de aplicación DB2.

El primer parámetro, $1, especifica el nombre del archivo fuente. Éste es el único parámetro necesario para los programas que no contienen SQL incorporado. Para

crear programas de SQL incorporado es necesaria una conexión con la base de datos, por lo que también se proporcionan otros tres parámetros opcionales: el segundo parámetro, $2, especifica el nombre de la base de datos con la que se desea conectar; el tercer parámetro, $3, especifica el ID de usuario correspondiente a la base de datos, y $4 especifica la contraseña.

Para un programa de SQL incorporado, bldapp pasa los parámetros al script de precompilación y vinculación, embprep. Si no se proporciona un nombre de base de datos, se utiliza la base de datos por omisión sample. Los parámetros de ID de usuario y contraseña sólo son necesarios si la instancia donde se crea el programa es diferente de la instancia donde reside la base de datos.

Para crear el programa de ejemplo de SQL no incorporado, client, a partir del archivo fuente client.cbl, especifique:

```
bldapp client
```

El resultado es un archivo ejecutable, client. Para ejecutar el archivo ejecutable sobre la base de datos sample, especifique:

```
client
```

Creación y ejecución de aplicaciones de SQL incorporado

- A partir del archivo fuente updat.sqb, puede crear la aplicación de SQL incorporado, updat, de tres maneras:
  1. Si conecta con la base de datos "sample" en la misma instancia, especifique:
     ```
     bldapp updat
     ```
  2. Si conecta con otra base de datos en la misma instancia, especifique también el nombre de la base de datos:
     ```
     bldapp updat basedatos
     ```
  3. Si conecta con una base de datos de otra instancia, especifique también el ID de usuario y la contraseña de la instancia de base de datos:
     ```
     bldapp updat basedatos IDusuario contraseña
     ```
     El resultado es un archivo ejecutable, updat.
- Existen tres maneras de ejecutar esta aplicación de SQL incorporado:
  1. Si desea acceder a la base de datos sample en la misma instancia, especifique simplemente el nombre del ejecutable:
     ```
     updat
     ```
  2. Si desea acceder a otra base de datos en la misma instancia, especifique el nombre del ejecutable y el nombre de la base de datos:
     ```
     updat basedatos
     ```
  3. Si desea acceder a una base de datos de otra instancia, especifique el nombre del ejecutable, el nombre de la base de datos, y el ID de usuario y la contraseña de la instancia de base de datos:
     ```
     bldapp updat basedatos IDusuario contraseña
     ```

Creación de aplicaciones IBM COBOL en Windows

DB2 proporciona scripts de creación para compilar y enlazar programas de la API de DB2 y de SQL incorporado. Estos archivos residen en el directorio sqllib\samples\cobol, junto con programas de ejemplo que se pueden crear a partir de esos archivos.

DB2 da soporte a dos precompiladores para crear aplicaciones IBM COBOL en Windows, el precompilador de DB2 y el precompilador de IBM COBOL. El valor

por omisión es el precompilador de DB2. El precompilador de IBM COBOL se puede seleccionar descomentando la línea apropiada del archivo de proceso por lotes que se esté utilizando. La precompilación con IBM COBOL la realiza el propio compilador, utilizando opciones de precompilación específicas.

El archivo de proceso por lotes `bldapp.bat` contiene los mandatos para crear programas de aplicación DB2. Utiliza un máximo de cuatro parámetros como entrada, que están representados dentro del archivo de proceso por lotes por las variables `%1`, `%2`, `%3` y `%4`.

El primer parámetro, `%1`, especifica el nombre del archivo fuente. Éste es el único parámetro necesario para los programas que no contienen SQL incorporado. Para crear programas de SQL incorporado es necesaria una conexión con la base de datos, por lo que también se proporcionan otros tres parámetros: el segundo parámetro, `%2`, especifica el nombre de la base de datos con la que se desea conectar; el tercer parámetro, `%3`, especifica el ID de usuario correspondiente a la base de datos, y `%4` especifica la contraseña.

Para un programa de SQL incorporado que utiliza el precompilador de DB2 por omisión, `bldapp.bat` pasa los parámetros al archivo de precompilación y vinculación, `embprep.bat`.

Para un programa de SQL incorporado que utiliza el precompilador de IBM COBOL, `bldrtn.bat` copia el archivo fuente `.sqb` en un archivo fuente `.cbl`. El compilador realiza la precompilación sobre el archivo fuente `.cbl` con opciones de precompilación específicas.

Para cualquiera de los precompiladores, si no se proporciona un nombre de base de datos, se utiliza la base de datos por omisión `sample`. Los parámetros de ID de usuario y contraseña sólo son necesarios si la instancia donde se crea el programa es diferente de la instancia donde reside la base de datos.

Los ejemplos siguientes muestran cómo crear y ejecutar programas de la API de DB2 y aplicaciones de SQL incorporado.

Para crear el programa de ejemplo de SQL no incorporado, `client`, a partir del archivo fuente `client.cbl`, especifique:

```
bldapp client
```

El resultado es un archivo ejecutable, `client.exe`. Para ejecutar el archivo ejecutable sobre la base de datos `sample`, especifique el nombre del ejecutable (sin la extensión):

```
client
```

Creación y ejecución de aplicaciones de SQL incorporado

- A partir del archivo fuente `updat.sqb`, puede crear la aplicación de SQL incorporado, `updat`, de tres maneras:
  1. Si conecta con la base de datos ″sample″ en la misma instancia, especifique:
     ```
     bldapp updat
     ```
  2. Si conecta con otra base de datos en la misma instancia, especifique también el nombre de la base de datos:
     ```
     bldapp updat basedatos
     ```
  3. Si conecta con una base de datos de otra instancia, especifique también el ID de usuario y la contraseña de la instancia de base de datos:

```
bldapp updat basedatos IDusuario contraseña
```

El resultado es un archivo ejecutable, `updat`.

- Existen tres maneras de ejecutar esta aplicación de SQL incorporado:
  1. Si desea acceder a la base de datos `sample` en la misma instancia, especifique simplemente el nombre del ejecutable:
     ```
     updat
     ```
  2. Si desea acceder a otra base de datos en la misma instancia, especifique el nombre del ejecutable y el nombre de la base de datos:
     ```
     updat basedatos
     ```
  3. Si desea acceder a una base de datos de otra instancia, especifique el nombre del ejecutable, el nombre de la base de datos, y el ID de usuario y la contraseña de la instancia de base de datos:
     ```
     bldapp updat basedatos IDusuario contraseña
     ```

Creación de aplicaciones Micro Focus COBOL en Windows

DB2 proporciona scripts de creación para compilar y enlazar programas de la API de DB2 y de SQL incorporado. Estos archivos residen en el directorio `sqllib\samples\cobol_mf`, junto con programas de ejemplo que se pueden crear a partir de esos archivos.

El archivo de proceso por lotes `bldapp.bat` contiene los mandatos para crear un programa de aplicación DB2. Utiliza un máximo de cuatro parámetros como entrada, que están representados dentro del archivo de proceso por lotes por las variables `%1`, `%2`, `%3` y `%4`.

El primer parámetro, `%1`, especifica el nombre del archivo fuente. Éste es el único parámetro necesario para los programas que no contienen SQL incorporado. Para crear programas de SQL incorporado es necesaria una conexión con la base de datos, por lo que también se proporcionan otros tres parámetros: el segundo parámetro, `%2`, especifica el nombre de la base de datos con la que se desea conectar; el tercer parámetro, `%3`, especifica el ID de usuario correspondiente a la base de datos, y `%4` especifica la contraseña.

Para un programa de SQL incorporado, `bldapp` pasa los parámetros al archivo de precompilación y vinculación, `embprep.bat`. Si no se proporciona un nombre de base de datos, se utiliza la base de datos por omisión `sample`. Los parámetros de ID de usuario y contraseña sólo son necesarios si la instancia donde se crea el programa es diferente de la instancia donde reside la base de datos.

Los ejemplos siguientes muestran cómo crear y ejecutar programas de la API de DB2 y aplicaciones de SQL incorporado.

Para crear el programa de ejemplo de SQL no incorporado, `client`, a partir del archivo fuente `client.cbl`, especifique:

```
bldapp client
```

El resultado es un archivo ejecutable, `client.exe`. Para ejecutar el archivo ejecutable sobre la base de datos `sample`, especifique el nombre del ejecutable (sin la extensión):

```
client
```

Creación y ejecución de aplicaciones de SQL incorporado

- A partir del archivo fuente `updat.sqb`, puede crear la aplicación de SQL incorporado, `updat`, de tres maneras:
  1. Si conecta con la base de datos "sample" en la misma instancia, especifique:
     ```
     bldapp updat
     ```
  2. Si conecta con otra base de datos en la misma instancia, especifique también el nombre de la base de datos:
     ```
     bldapp updat basedatos
     ```
  3. Si conecta con una base de datos de otra instancia, especifique también el ID de usuario y la contraseña de la instancia de base de datos:
     ```
     bldapp updat basedatos IDusuario contraseña
     ```

     El resultado es un archivo ejecutable, `updat.exe`.
- Existen tres maneras de ejecutar esta aplicación de SQL incorporado:
  1. Si accede a la base de datos `sample` en la misma instancia, especifique simplemente el nombre del ejecutable (sin la extensión):
     ```
     updat
     ```
  2. Si desea acceder a otra base de datos en la misma instancia, especifique el nombre del ejecutable y el nombre de la base de datos:
     ```
     updat basedatos
     ```
  3. Si desea acceder a una base de datos de otra instancia, especifique el nombre del ejecutable, el nombre de la base de datos, y el ID de usuario y la contraseña de la instancia de base de datos:
     ```
     bldapp updat basedatos IDusuario contraseña
     ```

Creación y ejecución de aplicaciones de SQL incorporado escritas en 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.

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.

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 nombre_archivo
  ```
- 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 REXX/6000 está en el directorio /usr/bin, incluya lo 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:/$DB2PATH/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 de archivos de vinculación son:

DB2ARXCS.BND
: Soporta el nivel de aislamiento de estabilidad del cursor.

DB2ARXRR.BND
: Soporta el nivel de aislamiento de lectura repetitiva.

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 o System i. 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.

Creación de aplicaciones Object REXX en Windows

Object REXX es un versión del lenguaje REXX orientada a objetos. Se han añadido extensiones orientadas a objetos al REXX estándar, pero no se han modificado su funciones e instrucciones existentes. El programa intérprete de Object REXX es una versión mejorada de su predecesor, e incluye soporte adicional para:

- Clases, objetos y métodos
- Gestión de mensajes y polimorfismo
- Herencia simple y múltiple

Object REXX es totalmente compatible con el REXX estándar. En esta sección, siempre que se utiliza el término REXX se hace referencia a todas las versiones de REXX, incluido Object REXX.

No es necesario precompilar ni enlazar los programas REXX.

En Windows, no es necesario que los programas REXX comiencen con un comentario. Sin embargo, por razones de portabilidad, es recomendable que en cada programa REXX coloque un comentario que comience en la primera columna de la primera línea. Esto permitirá diferenciar el programa respecto de un mandato de proceso por lotes cuando se ejecute en otras plataformas:

```
/* Coloque aquí un comentario cualquiera. */
```

Los programas REXX de ejemplo están contenidos en el directorio `sqllib\samples\rexx`.

Para ejecutar el programa REXX de ejemplo `updat`, especifique:

```
rexx updat.cmd
```

Creación de aplicaciones de SQL incorporado desde la línea de mandatos

Para crear de aplicaciones de SQL incorporado desde la línea de mandatos incluye, siga estos pasos:

1. Precompile la aplicación emitiendo el mandato PRECOMPILE
2. Si ha creado un archivo de vinculación, vincule dicho archivo a una base de datos para crear un paquete de aplicación emitiendo el mandato BIND.
3. Compile la fuente de la aplicación modificada y los archivos fuente que no contienen SQL incorporado para crear un archivo de objeto de aplicación (un archivo .obj).
4. Enlace los archivos de objeto de aplicación con DB2 y con bibliotecas del lenguaje principal para crear un programa ejecutable mediante el mandato link.

Creación de aplicaciones de SQL incorporado escritas en C o C++ (Windows)

Después de escribir el archivo fuente, tiene que crear la aplicación de SQL incorporado. Algunos pasos del proceso de creación dependen del compilador que utilice. Los ejemplos que se proporcionan con casa paso del procedimiento muestran cómo crear una aplicación denominada myapp con un compilador Microsoft Visual Studio 6.0, que es un compilador de C. Puede ejecutar cada uno de los pasos del procedimiento de forma individual o puede ejecutar los pasos juntos dentro de un archivo de proceso por lotes desde un indicador de la ventana de mandatos de DB2. Para ver un ejemplo de un archivo de proceso por lotes que se puede utilizar para crear aplicaciones de ejemplo de SQL incorporado en el directorio %DB2PATH%\SQLLIB\samples\c\, consulte el archivo %DB2PATH%\SQLLIB\samples\c\bldapp.bat. Este archivo de proceso por lotes

llama a otro archivo de proceso por lotes, %DB2PATH%\SQLLIB\samples\c\ embprep.bat, para precompilar la aplicación y vincular la aplicación a una base de datos.

- Una conexión de base de datos activa
- Un archivo de código fuente de aplicación con la extensión .sqc en C o .sqx en C++ y que contenga SQL incorporado
- Un compilador de C o C++ soportado
- Las autorizaciones o privilegios necesarios para ejecutar el mandato PRECOMPILE y el mandato BIND

1. Precompile la aplicación emitiendo el mandato PRECOMPILE. Por ejemplo:

   ```
   Aplicación C: db2 PRECOMPILE myapp.sqc BINDFILE
   Aplicación C++: db2 PRECOMPILE myapp.sqx BINDFILE
   ```

   El mandato PRECOMPILE genera un archivo .c o .C, que contiene un formato modificado del código fuente en un archivo .sqc o .sqC, un paquete de aplicación. Si utiliza la opción BINDFILE, el mandato PRECOMPILE genera un archivo de vinculación. En el ejemplo anterior, el archivo de vinculación se llamaría myapp.bnd.
2. Si ha creado un archivo de vinculación, vincule dicho archivo a una base de datos para crear un paquete de aplicación emitiendo el mandato BIND. Por ejemplo:

   ```
   db2 bind myapp.bnd
   ```

   El mandato BIND asocia el paquete de aplicación a la base de datos y almacena el paquete dentro de la misma.
3. Compile la fuente de la aplicación modificada y los archivos fuente que no contienen SQL incorporado para crear un archivo de objeto de aplicación (un archivo .obj). Por ejemplo:

   ```
   Aplicación C: cl -Zi -Od -c -W2 -DWIN32 myapp.c
   Aplicación C++: cl -Zi -Od -c -W2 -DWIN32 myapp.cxx
   ```
4. Enlace los archivos de objeto de aplicación con DB2 y con bibliotecas del lenguaje principal para crear un programa ejecutable mediante el mandato link. Por ejemplo:

   ```
   link -debug -out:myapp.exe myapp.obj
   ```

Siguiente tarea

A continuación puede realizar cualquiera de las siguientes tareas:

- Depuración de errores en tiempo de compilación de la aplicación de SQL incorporado
- Ajuste del rendimiento de aplicaciones de SQL incorporado
- Despliegue de la aplicación ejecutable, myapp.exe
- Ejecución de aplicaciones de SQL incorporado

Capítulo 7. Despliegue y ejecución de aplicaciones de SQL incorporado

Las aplicaciones de SQL incorporado son portables y pueden colocarse en máquinas remotas. Puede compilar la aplicación en una ubicación y ejecutar el paquete en otra máquina para utilizar la base de datos en la máquina más nueva.

Restricciones de enlazar a libdb2.so

En algunas distribuciones de Linux, el rpm de desarrollo de libc viene con la biblioteca

`/usr/lib/libdb2.so`

o

`/usr/lib64/libdb2.so`

. Esta biblioteca se utiliza para la implementación Berkeley DB de Sleepycat Software y no está asociada con sistemas de base de datos IBM DB2.

Si no tiene pensado utilizar Berkeley DB, puede redenominar o suprimir estos archivos de biblioteca permanentemente en los sistemas.

Si desea utilizar Berkeley DB, puede redenominar la carpeta que contiene estos archivos de biblioteca y modificar la variable de entorno para que señale a la nueva carpeta.

Parte 2. Apéndices

Apéndice A. Visión general de la información técnica de DB2

La información técnica de DB2 está disponible a través de las herramientas y los métodos siguientes:

- Centro de información de DB2
  - Temas (Tareas, concepto y temas de consulta)
  - Ayuda para herramientas de DB2
  - Programas de ejemplo
  - Guías de aprendizaje
- Manuales de DB2
  - Archivos PDF (descargables)
  - Archivos PDF (desde el DVD con PDF de DB2)
  - Manuales en copia impresa
- Ayuda de línea de mandatos
  - Ayuda de mandatos
  - Ayuda de mensajes

Nota: Los temas del Centro de información de DB2 se actualizan con más frecuencia que los manuales en PDF o impresos. Para obtener la información más actualizada, instale las actualizaciones de la documentación cuando estén disponibles, o consulte el Centro de información de DB2 en ibm.com.

Puede acceder a información técnica adicional de DB2 como, por ejemplo, notas técnicas, documentos técnicos y publicaciones IBM Redbooks en línea, en el sitio ibm.com. Acceda al sitio de la biblioteca de software de gestión de información de DB2 en http://www.ibm.com/software/data/sw-library/.

Comentarios sobre la documentación

Agradecemos los comentarios sobre la documentación de DB2. Si tiene sugerencias sobre cómo podemos mejorar la documentación de DB2, envíe un correo electrónico a db2docs@ca.ibm.com. El personal encargado de la documentación de DB2 lee todos los comentarios de los usuarios, pero no puede responderlos directamente. Proporcione ejemplos específicos siempre que sea posible de manera que podamos comprender mejor sus problemas. Si realiza comentarios sobre un tema o archivo de ayuda determinado, incluya el título del tema y el URL.

No utilice esta dirección de correo electrónico para contactar con el Soporte al cliente de DB2. Si tiene un problema técnico de DB2 que no está tratado por la documentación, consulte al centro local de servicio técnico de IBM para obtener ayuda.

Biblioteca técnica de DB2 en copia impresa o en formato PDF

Las tablas siguientes describen la biblioteca de DB2 que está disponible en el Centro de publicaciones de IBM en www.ibm.com/shop/publications/order. Los manuales de DB2 Versión 9.5 en inglés en formato PDF y las versiones traducidas se pueden descargar del sitio www.ibm.com/support/docview.wss?rs=71&uid=swg2700947.

Aunque las tablas identifican los manuales en copia impresa disponibles, puede que dichos manuales no estén disponibles en su país o región.

El número de documento se incrementa cada vez que se actualiza un manual. Asegúrese de que lee la versión más reciente de los manuales, tal como aparece a continuación:

Nota: El Centro de información de DB2 se actualiza con más frecuencia que los manuales en PDF o impresos.

Tabla 23. Información técnica de DB2

Nombre	Número de documento	Copia impresa disponible
Consulta de las API administrativas	SC11-3505-01	Sí
Rutinas y vistas administrativas	SC11-3507-01	No
Call Level Interface Guide and Reference, Volume 1	SC23-5844-01	Sí
Call Level Interface Guide and Reference, Volume 2	SC23-5845-01	Sí
Consulta de mandatos	SC11-3506-01	Sí
Data Movement Utilities Guide and Reference	SC23-5847-01	Sí
Data Recovery and High Availability Guide and Reference	SC23-5848-01	Sí
Data Servers, Databases, and Database Objects Guide	SC23-5849-01	Sí
Database Security Guide	SC23-5850-01	Sí
Desarrollo de aplicaciones ADO.NET y OLE DB	SC11-3499-01	Sí
Desarrollo de aplicaciones de SQL incorporado	SC11-3500-01	Sí
Desarrollo de aplicaciones Java	SC11-3501-01	Sí
Desarrollo de aplicaciones Perl y PHP	SC11-3502-01	No
Desarrollo de rutinas definidas por el usuario (SQL y externas)	SC11-3503-01	Sí
Iniciación al desarrollo de aplicaciones de bases de datos	GC11-3504-01	Sí
Iniciación a la instalación y administración de DB2 en Linux y Windows	GC11-3511-01	Sí
Internationalization Guide	SC23-5858-01	Sí
Consulta de mensajes, Volumen 1	GI11-7823-00	No
Consulta de mensajes, Volumen 2	GI11-7824-00	No
Guía de migración	GC11-3510-01	Sí
Net Search Extender Guía de administración y del usuario	SC11-3615-01	Sí
Partitioning and Clustering Guide	SC23-5860-01	Sí

Tabla 23. Información técnica de DB2 (continuación)

Nombre	Número de documento	Copia impresa disponible
Query Patroller Administration and User's Guide	SC23-8507-00	Sí
Guía rápida de iniciación para clientes IBM Data Server	GC11-3513-01	No
Guía rápida de iniciación para servidores DB2	GC11-3512-01	Sí
Spatial Extender and Geodetic Data Management Feature Guía del usuario y manual de consulta	SC11-3614-01	Sí
Consulta de SQL, Volumen 1	SC11-3508-01	Sí
Consulta de SQL, Volumen 2	SC11-3509-01	Sí
System Monitor Guide and Reference	SC23-5865-01	Sí
Troubleshooting Guide	GI11-7857-01	No
Tuning Database Performance	SC23-5867-01	Sí
Guía de aprendizaje de Visual Explain	SC11-3518-00	No
Novedades	SC11-3517-01	Sí
Workload Manager Guide and Reference	SC23-5870-01	Sí
pureXML Guide	SC23-5871-01	Sí
XQuery Reference	SC23-5872-01	No

Tabla 24. Información técnica específica de DB2 Connect

Nombre	Número de documento	Copia impresa disponible
Guía rápida de iniciación para DB2 Connect Personal Edition	GC11-3515-01	Sí
Guía rápida de iniciación para servidores DB2 Connect	GC11-3516-01	Sí
Guía del usuario de DB2 Connect	SC11-3514-01	Sí

Tabla 25. Información técnica de Information Integration

Nombre	Número de documento	Copia impresa disponible
Information Integration: Administration Guide for Federated Systems	SC19-1020-01	Sí
Information Integration: ASNCLP Program Reference for Replication and Event Publishing	SC19-1018-02	Sí
Information Integration: Configuration Guide for Federated Data Sources	SC19-1034-01	No
Information Integration: SQL Replication Guide and Reference	SC19-1030-01	Sí

Tabla 25. Información técnica de Information Integration (continuación)

Nombre	Número de documento	Copia impresa disponible
Information Integration: Introduction to Replication and Event Publishing	SC19-1028-01	Sí

Pedido de manuales de DB2 en copia impresa

Si necesita manuales de DB2 en copia impresa, puede comprarlos en línea en varios países o regiones, pero no en todos. Siempre puede hacer pedidos de manuales de DB2 en copia impresa a través del representante local de IBM. Recuerde que algunas publicaciones en copia software del DVD *Documentación en PDF de DB2* no están disponibles en copia impresa. Por ejemplo, no está disponible la publicación *Consulta de mensajes de DB2* en copia impresa.

Las versiones impresas de muchas de las publicaciones de DB2 disponibles en el DVD de Documentación en PDF de DB2 se pueden solicitar a IBM por una cantidad. Dependiendo desde dónde realice el pedido, podrá solicitar manuales en línea, desde el Centro de publicaciones de IBM. Si la realización de pedidos en línea no está disponible en su país o región, siempre puede hacer pedidos de manuales de DB2 en copia impresa al representante local de IBM. Tenga en cuenta que no todas las publicaciones del DVD de Documentación en PDF de DB2 están disponibles en copia impresa.

Nota: La documentación más actualizada y completa de DB2 se conserva en el Centro de información de DB2 en http://publib.boulder.ibm.com/infocenter/db2luw/v9r5.

Para hacer pedidos de manuales de DB2 en copia impresa:

- Para averiguar si puede hacer pedidos de manuales de DB2 en copia impresa en línea en su país o región, consulte el Centro de publicaciones de IBM en el sitio http://www.ibm.com/shop/publications/order. Debe seleccionar un país, región o idioma para poder acceder a la información sobre pedidos de publicaciones y, a continuación, seguir las instrucciones sobre pedidos para su localidad.
- Para hacer pedidos de manuales de DB2 en copia impresa a través del representante local de IBM:
  1. Localice la información de contacto de su representante local desde uno de los siguientes sitios Web:
     - El directorio de IBM de contactos en todo el mundo en el sitio www.ibm.com/planetwide
     - El sitio Web de publicaciones de IBM en el sitio http://www.ibm.com/shop/publications/order. Tendrá que seleccionar su país, región o idioma para acceder a la página de presentación de las publicaciones apropiadas para su localidad. Desde esta página, siga el enlace "Acerca de este sitio".
  2. Cuando llame, indique que desea hacer un pedido de una publicación de DB2.
  3. Proporcione al representante los títulos y números de documento de las publicaciones que desee solicitar. Si desea consultar los títulos y los números de documento, consulte el apartado “Biblioteca técnica de DB2 en copia impresa o en formato PDF” en la página 213.

Visualización de la ayuda para estados de SQL desde el procesador de línea de mandatos

DB2 devuelve un valor de SQLSTATE para las condiciones que pueden ser el resultado de una sentencia de SQL. La ayuda de SQLSTATE explica los significados de los estados de SQL y los códigos de las clases de estados de SQL.

Para invocar la ayuda para estados de SQL, abra el procesador de línea de mandatos y entre:

```
? sqlstate o ? código de clase
```

donde *sqlstate* representa un estado de SQL válido de cinco dígitos y *código de clase* representa los dos primeros dígitos del estado de SQL.

Por ejemplo, `? 08003` visualiza la ayuda para el estado de SQL 08003, y `? 08` visualiza la ayuda para el código de clase 08.

Acceso a diferentes versiones del Centro de información de DB2

Para los temas de DB2 Version 9.5, el URL del Centro de información de DB2 es http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/

Para los temas de DB2 Version 9, el URL del Centro de información de DB2 es http://publib.boulder.ibm.com/infocenter/db2luw/v9/

Para los temas de DB2 Version 8, vaya al URL del Centro de información de la Versión 8 en el sitio: http://publib.boulder.ibm.com/infocenter/db2luw/v8/

Visualización de temas en su idioma preferido en el Centro de información de DB2

El Centro de información de DB2 intenta visualizar los temas en el idioma especificado en las preferencias del navegador. Si un tema no se ha traducido al idioma preferido, el Centro de información de DB2 visualiza dicho tema en inglés.

- Para visualizar temas en su idioma preferido en el navegador Internet Explorer:
  1. En Internet Explorer, pulse en el botón **Herramientas** —> **Opciones de Internet** —> **Idiomas...**. Se abrirá la ventana Preferencias de idioma.
  2. Asegúrese de que su idioma preferido esté especificado como la primera entrada de la lista de idiomas.
     - Para añadir un nuevo idioma a la lista, pulse el botón **Agregar...**.

       Nota: La adición de un idioma no garantiza que el sistema tenga los fonts necesarios para visualizar los temas en el idioma preferido.
     - Para mover un idioma hacia el principio de la lista, seleccione el idioma y pulse el botón **Subir** hasta que el idioma esté en primer lugar en la lista de idiomas.
  3. Borre la antememoria del navegador y, a continuación, renueve la página para visualizar el Centro de información de DB2 en su idioma preferido.
- Para visualizar temas en su idioma preferido en un navegador Firefox o Mozilla:
  1. Seleccione el botón en la sección **Idiomas** del diálogo **Herramientas** —> **Opciones** —> **Avanzado**. Se visualizará el panel Idiomas en la ventana Preferencias.

2. Asegúrese de que su idioma preferido esté especificado como la primera entrada de la lista de idiomas.
   - Para añadir un nuevo idioma a la lista, pulse el botón **Añadir...** a fin de seleccionar un idioma en la ventana Añadir idiomas.
   - Para mover un idioma hacia el principio de la lista, seleccione el idioma y pulse el botón **Subir** hasta que el idioma esté en primer lugar en la lista de idiomas.
3. Borre la antememoria del navegador y, a continuación, renueve la página para visualizar el Centro de información de DB2 en su idioma preferido.

En algunas combinaciones de navegador y sistema operativo, puede que también tenga que cambiar los valores regionales del sistema operativo al entorno local y al idioma de su elección.

Actualización del Centro de información de DB2 instalado en el sistema o en el servidor de intranet

Si ha instalado localmente el Centro de información de DB2, puede obtener las actualizaciones de la documentación de IBM e instalarlas.

Para actualizar el Centro de información de DB2 instalado localmente es preciso que:

1. Detenga el Centro de información de DB2 en el sistema, y reinicie el Centro de información en modalidad autónoma. La ejecución del Centro de información en modalidad autónoma impide que otros usuarios de la red accedan al Centro de información y permite al usuario aplicar las actualizaciones. Los Centros de información no administrativos y no root de DB2 se ejecutan siempre en modalidad autónoma.
2. Utilice la función Actualizar para ver qué actualizaciones están disponibles. Si hay actualizaciones que desee instalar, puede utilizar la función Actualizar para obtenerlas e instalarlas

   Nota: Si su entorno requiere la instalación de actualizaciones del Centro de información de DB2 en una máquina no conectada a Internet, debe duplicar el sitio de actualizaciones en un sistema de archivos local utilizando una máquina que esté conectada a Internet y tenga instalado el Centro de información de DB2. Si muchos usuarios en la red van a instalar las actualizaciones de la documentación, puede reducir el tiempo necesario para realizar las actualizaciones duplicando también el sitio de actualizaciones localmente y creando un proxy para el sitio de actualizaciones.
   Si hay paquetes de actualización disponibles, utilice la característica Actualizar para obtener los paquetes. Sin embargo, la característica Actualizar sólo está disponible en modalidad autónoma.
3. Detenga el Centro de información autónomo y reinicie el Centro de información de DB2 en su equipo.

Nota: En Windows Vista, los mandatos listados más abajo se deben ejecutar como administrador. Para iniciar un indicador de mandatos o una herramienta gráfica con privilegios de administrador completos, pulse con el botón derecho del ratón el atajo y, a continuación, seleccione **Ejecutar como administrador**.

Para actualizar el Centro de información de DB2 instalado en el sistema o en el servidor de Intranet:

1. Detenga el Centro de información de DB2.

- En Windows, pulse **Inicio** → **Panel de control** → **Herramientas administrativas** → **Servicios**. A continuación, pulse con el botón derecho del ratón en el servicio **Centro de información de DB2** y seleccione **Detener**.
- En Linux, especifique el mandato siguiente:

  ```
  /etc/init.d/db2icdv95 stop
  ```

2. Inicie el Centro de información en modalidad autónoma.
   - En Windows:
     a. Abra una ventana de mandatos.
     b. Navegue hasta la vía de acceso en la que está instalado el Centro de información. De forma predeterminada, el Centro de información de DB2 se instala en el directorio <Archivos de programa>\IBM\Centro de información de DB2\Versión 9.5, siendo `<Archivos de programa>` la ubicación del directorio Archivos de programa.
     c. Navegue desde el directorio de instalación al directorio doc\bin.
     d. Ejecute el archivo help_start.bat:

        ```
        help_start.bat
        ```

   - En Linux:
     a. Navegue hasta la vía de acceso en la que está instalado el Centro de información. De forma predeterminada, el Centro de información de DB2 se instala en el directorio /opt/ibm/db2ic/V9.5.
     b. Navegue desde el directorio de instalación al directorio doc/bin.
     c. Ejecute el script help_start:

        ```
        help_start
        ```

   Se inicia el navegador Web por omisión del sistema para visualizar el Centro de información autónomo.
3. Pulse en el botón **Actualizar** (). En la derecha del panel del Centro de información, pulse en **Buscar actualizaciones**. Se visualiza una lista de actualizaciones para la documentación existente.
4. Para iniciar el proceso de instalación, compruebe las selecciones que desee instalar y, a continuación, pulse **Instalar actualizaciones**.
5. Cuando finalice el proceso de instalación, pulse **Finalizar**.
6. Detenga el Centro de información autónomo:
   - En Windows, navegue hasta el directorio doc\bin del directorio de instalación y ejecute el archivo help_end.bat:

     ```
     help_end.bat
     ```

     Nota: El archivo help_end de proceso por lotes contiene los mandatos necesarios para concluir sin peligro los procesos que se iniciaron mediante el archivo help_start de proceso por lotes. No utilice `Control-C` ni ningún otro método para concluir help_start.bat.
   - En Linux, navegue hasta el directorio de instalación doc/bin y ejecute el script help_end:

     ```
     help_end
     ```

     Nota: El script help_end contiene los mandatos necesarios para concluir sin peligro los procesos que se iniciaron mediante el script help_start. No utilice ningún otro método para concluir el script help_start.
7. Reinicie el Centro de información de DB2:

- En Windows, pulse **Inicio → Panel de control → Herramientas administrativas → Servicios**. A continuación, pulse con el botón derecho del ratón en el servicio **Centro de información de DB2** y seleccione **Iniciar**.
- En Linux, especifique el mandato siguiente:

  ```
  /etc/init.d/db2icdv95 start
  ```

El Centro de información de DB2 actualizado visualiza los temas nuevos y actualizados.

Guías de aprendizaje de DB2

Las guías de aprendizaje de DB2 le ayudan a conocer diversos aspectos de productos DB2. Se proporcionan instrucciones paso a paso a través de lecciones.

Antes de comenzar

Puede ver la versión XHTML de la guía de aprendizaje desde el Centro de información en el sitio http://publib.boulder.ibm.com/infocenter/db2help/.

Algunas lecciones utilizan datos o código de ejemplo. Consulte la guía de aprendizaje para obtener una descripción de los prerrequisitos para las tareas específicas.

Guías de aprendizaje de DB2

Para ver la guía de aprendizaje, pulse el título.

"pureXML" en *pureXML Guide*
: Configure una base de datos DB2 para almacenar datos XML y realizar operaciones básicas con el almacén de datos XML nativos.

"Visual Explain" en *Guía de aprendizaje de Visual Explain*
: Analizar, optimizar y ajustar sentencias de SQL para obtener un mejor rendimiento al utilizar Visual Explain.

Información de resolución de problemas de DB2

Existe una gran variedad de información para la resolución y determinación de problemas para ayudarle en la utilización de productos DB2.

Documentación de DB2
: Puede encontrar información sobre la resolución de problemas en la publicación DB2 Troubleshooting Guide o en la sección Soporte y resolución de problemas del Centro de información de DB2. En ellas encontrará información sobre cómo aislar e identificar problemas utilizando herramientas y programas de utilidad de diagnóstico de DB2, soluciones a algunos de los problemas más habituales y otros consejos sobre cómo solucionar problemas que podría encontrar en los productos DB2.

Sitio web de soporte técnico de DB2
: Consulte el sitio Web de soporte técnico de DB2 si tiene problemas y desea obtener ayuda para encontrar las causas y soluciones posibles. El sitio de soporte técnico tiene enlaces a las publicaciones más recientes de DB2, notas técnicas, Informes autorizados de análisis del programa (APAR o arreglos de defectos), fixpacks y otros recursos. Puede buscar en esta base de conocimiento para encontrar posibles soluciones a los problemas.

Acceda al sitio Web de soporte técnico de DB2 en el sitio http://www.ibm.com/software/data/db2/udb/support.html

Términos y condiciones

Los permisos para utilizar estas publicaciones se otorgan sujetos a los siguientes términos y condiciones.

Uso personal: Puede reproducir estas publicaciones para su uso personal, no comercial, siempre y cuando se mantengan los avisos sobre la propiedad. No puede distribuir, visualizar o realizar trabajos derivados de estas publicaciones, o de partes de las mismas, sin el consentimiento expreso de IBM.

Uso comercial: Puede reproducir, distribuir y visualizar estas publicaciones únicamente dentro de su empresa, siempre y cuando se mantengan todos los avisos sobre la propiedad. No puede realizar trabajos derivados de estas publicaciones, ni reproducirlas, distribuirlas o visualizarlas, ni de partes de las mismas fuera de su empresa, sin el consentimiento expreso de IBM.

Excepto lo expresamente concedido en este permiso, no se conceden otros permisos, licencias ni derechos, explícitos o implícitos, sobre las publicaciones ni sobre ninguna información, datos, software u otra propiedad intelectual contenida en el mismo.

IBM se reserva el derecho de retirar los permisos aquí concedidos cuando, a su discreción, el uso de las publicaciones sea en detrimento de su interés o cuando, según determine IBM, las instrucciones anteriores no se cumplan correctamente.

No puede descargar, exportar ni volver a exportar esta información excepto en el caso de cumplimiento total con todas las leyes y regulaciones vigentes, incluyendo todas las leyes y regulaciones sobre exportación de los Estados Unidos.

IBM NO GARANTIZA EL CONTENIDO DE ESTAS PUBLICACIONES. LAS PUBLICACIONES SE PROPORCIONAN ″TAL CUAL″ Y SIN GARANTÍA DE NINGUNA CLASE, NI EXPLÍCITA NI IMPLÍCITA, INCLUYENDO PERO SIN LIMITARSE A LAS GARANTÍAS IMPLÍCITAS DE COMERCIALIZACIÓN, NO VULNERACIÓN E IDONEIDAD PARA UN FIN DETERMINADO.

Apéndice B. Avisos

Esta información ha sido desarrollada para productos y servicios que se ofrecen en Estados Unidos de América

Es posible que IBM no comercialice en otros países algunos productos, servicios o características descritos en este manual. Consulte al representante local de IBM para obtener información sobre los productos y servicios que actualmente pueden adquirirse en su zona. Cualquier referencia a un producto, programa o servicio de IBM no pretende afirmar ni implicar que sólo se pueda utilizar dicho producto, programa o servicio de IBM. En su lugar se puede utilizar cualquier producto, programa o servicio funcionalmente equivalente que no vulnere ninguno de los derechos de propiedad intelectual de IBM. Sin embargo, es responsabilidad del usuario evaluar y verificar el funcionamiento de cualquier producto, programa o servicio que no sea de IBM.

IBM puede tener patentes o solicitudes de patentes en tramitación que afecten al tema tratado en este documento. La posesión de este documento no confiere ninguna licencia sobre dichas patentes. Puede realizar consultas sobre licencias escribiendo a:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
EE.UU.

Para realizar consultas sobre licencias referentes a información de doble byte (DBCS), puede ponerse en contacto con el Departamento de Propiedad Intelectual de IBM de su país/región o escribir a:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokio 106, Japón

El párrafo siguiente no es aplicable al Reino Unido ni a ningún país/región en donde tales disposiciones sean incompatibles con la legislación local:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROPORCIONA ESTA PUBLICACIÓN “TAL CUAL”, SIN GARANTÍA DE NINGUNA CLASE, NI EXPLÍCITA NI IMPLÍCITA, INCLUIDAS, PERO SIN LIMITARSE A ELLAS, LAS GARANTÍAS IMPLÍCITAS DE NO VULNERACIÓN DE DERECHOS, COMERCIALIZACIÓN O IDONEIDAD PARA UN FIN DETERMINADO. Algunos estados no permiten la exclusión de garantías expresas o implícitas en determinadas transacciones, por lo que es posible que esta declaración no sea aplicable en su caso.

Esta publicación puede contener inexactitudes técnicas o errores tipográficos. Periódicamente se efectúan cambios en la información aquí contenida; dichos cambios se incorporarán a las nuevas ediciones de la publicación. IBM puede efectuar, en cualquier momento y sin previo aviso, mejoras y cambios en los productos y programas descritos en esta publicación.

Este documento puede proporcionar enlaces o referencias a sitios y recursos que no son de IBM. IBM no representa, no da garantías, ni se compromete con los recursos de terceros ni con los recursos que no son de IBM a los cuales se puede hacer referencia, acceder desde o enlazarse con desde este documento. Un enlace a un sitio que no es de IBM no implica que IBM apruebe el contenido o la utilización de dicho sitio Web o a su propietario. Además, IBM no forma parte ni es responsable de ninguna transacción que el usuario pueda realizar con terceros, aún cuando llegue a conocerlos (o utilice un enlace a ellos) desde un sitio de IBM. De acuerdo a esto, el usuario reconoce y acepta que IBM no es responsable de la disponibilidad de dichos recursos o sitios externos ni tampoco es responsable de ningún contenido, servicio, producto u otros materiales que estén o se encuentren disponibles desde dichos sitios o recursos. Cualquier software que proporcionen terceras partes, estarán sujetos a los términos y condiciones de licencia que acompañen al software.

IBM puede utilizar o distribuir cualquier información que se le facilite de la manera que considere adecuada, sin contraer por ello ninguna obligación con el remitente.

Los licenciatarios de este programa que deseen obtener información sobre él con el fin de habilitar: (i) el intercambio de información entre programas creados de forma independiente y otros programas (incluido éste) y (ii) el uso mutuo de la información intercambiada, deben ponerse en contacto con:

IBM Canada Limited
Office of the Lab Director
8200 Warden Avenue
Markham, Ontario
L6G 1C7
CANADÁ

Dicha información puede estar disponible, sujeta a los términos y condiciones apropiados, incluido en algunos casos el pago de una tarifa.

El programa bajo licencia descrito en este documento y todo el material bajo licencia asociado a él, los proporciona IBM según los términos del Acuerdo de Cliente de IBM, el Acuerdo Internacional de Programas Bajo Licencia de IBM o cualquier acuerdo equivalente entre el usuario e IBM.

Los datos de rendimiento contenidos en este documento se obtuvieron en un entorno controlado. Por lo tanto, los resultados obtenidos en otros entornos operativos pueden variar significativamente. Algunas mediciones pueden haberse realizado en sistemas experimentales y no es seguro que estas mediciones sean las mismas en los sistemas disponibles comercialmente. Además, algunas mediciones pueden haberse calculado mediante extrapolación. Los resultados reales pueden variar. Los usuarios del presente manual deben verificar los datos aplicables para su entorno específico.

La información referente a productos que no son de IBM se ha obtenido de los proveedores de esos productos, de sus anuncios publicados o de otras fuentes disponibles públicamente. IBM no ha probado esos productos y no puede confirmar la exactitud del rendimiento, la compatibilidad ni ninguna otra afirmación referente a productos que no son de IBM. Las preguntas sobre las prestaciones de productos que no son de IBM deben dirigirse a los proveedores de esos productos.

Todas las declaraciones de intenciones de IBM están sujetas a cambio o cancelación sin previo aviso, y sólo representan objetivos.

Este manual puede contener ejemplos de datos e informes que se utilizan en operaciones comerciales diarias. Para ilustrarlos de la forma más completa posible, los ejemplos incluyen nombres de personas, empresas, marcas y productos. Todos estos nombres son ficticios y cualquier similitud con nombres y direcciones utilizados por una empresa real es totalmente fortuita.

LICENCIA DE COPYRIGHT:

Este manual puede contener programas de aplicaciones de ejemplo escritos en lenguaje fuente, que muestran técnicas de programación en diversas plataformas operativas. Puede copiar, modificar y distribuir estos programas de ejemplo como desee, sin pago alguno a IBM con la intención de desarrollar, utilizar, comercializar o distribuir programas de aplicaciones de acuerdo con la interfaz de programación de aplicaciones correspondiente a la plataforma operativa para la que están escritos los programas de ejemplo. Estos ejemplos no se han probado exhaustivamente bajo todas las condiciones. Por lo tanto, IBM no puede asegurar ni implicar la fiabilidad, utilidad o función de estos programas.

Cada copia o parte de estos programas de ejemplo o cualquier trabajo derivado debe incluir una nota de copyright como la siguiente:

© (*nombre de la empresa*) (*año*). Partes de este código proceden de programas de ejemplo de IBM Corp. © Copyright IBM Corp. *_entre el o los años_*. Reservados todos los derechos.

Marcas registradas

Los siguientes términos son marcas registradas de International Business Machines Corporation en los EE.UU. y/o en otros países.

pureXML
MQSeries
DB2
System z9
AISPO
POWER
WebSphere
DB2 Connect
TXSeries
z/OS
System i
AS/400
IBM
zSeries
Rational
Tivoli
OS/400
Approach
pSeries
VisualAge
OpenPower
REXX
AIX
System z
Encina
OS/390
DB2 Universal Database
Redbooks
developerWorks
PowerPC
CICS
SQL/400
Lotus
HACMP
MVS
eServer
ibm.com
iSeries

Los siguientes términos son marcas registradas de otras empresas.

- Linux es una marca registrada de Linus Torvalds en los Estados Unidos y/o en otros países.
- Java y todas las marcas comerciales basadas en Java son marcas comerciales de Sun Microsystems, Inc. en los Estados Unidos y/o en otros países.
- UNIX es una marca registrada de The Open Group en los Estados Unidos y/o en otros países.
- Intel Xeon, Itanium y Pentium e Intel son marcas comerciales de Intel Corporation o de sus subsidiarias en los Estados Unidos y/o en otros países.
- Microsoft y Windows son marcas comerciales de Microsoft Corporation, Inc. en los Estados Unidos y/o en otros países.

Otros nombres de empresas, productos o servicios, pueden ser marcas registradas o marcas de servicio de otras empresas.

Índice

Caracteres Especiales

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X

SC11-3500-01

Spine information:

DB2 Versión 9.5 para Linux, UNIX y Windows

Desarrollo de aplicaciones de SQL incorporado