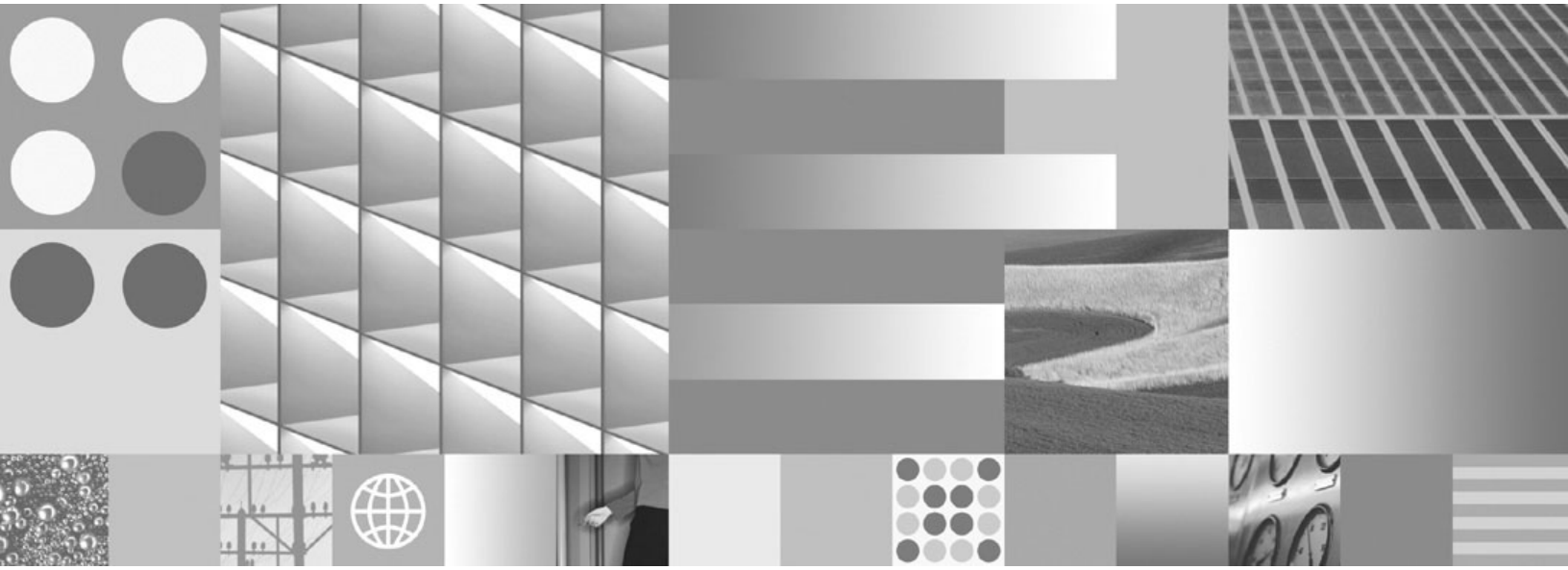
DB2 Versión 9.5
para Linux, UNIX y Windows

Versión 9 Release 5



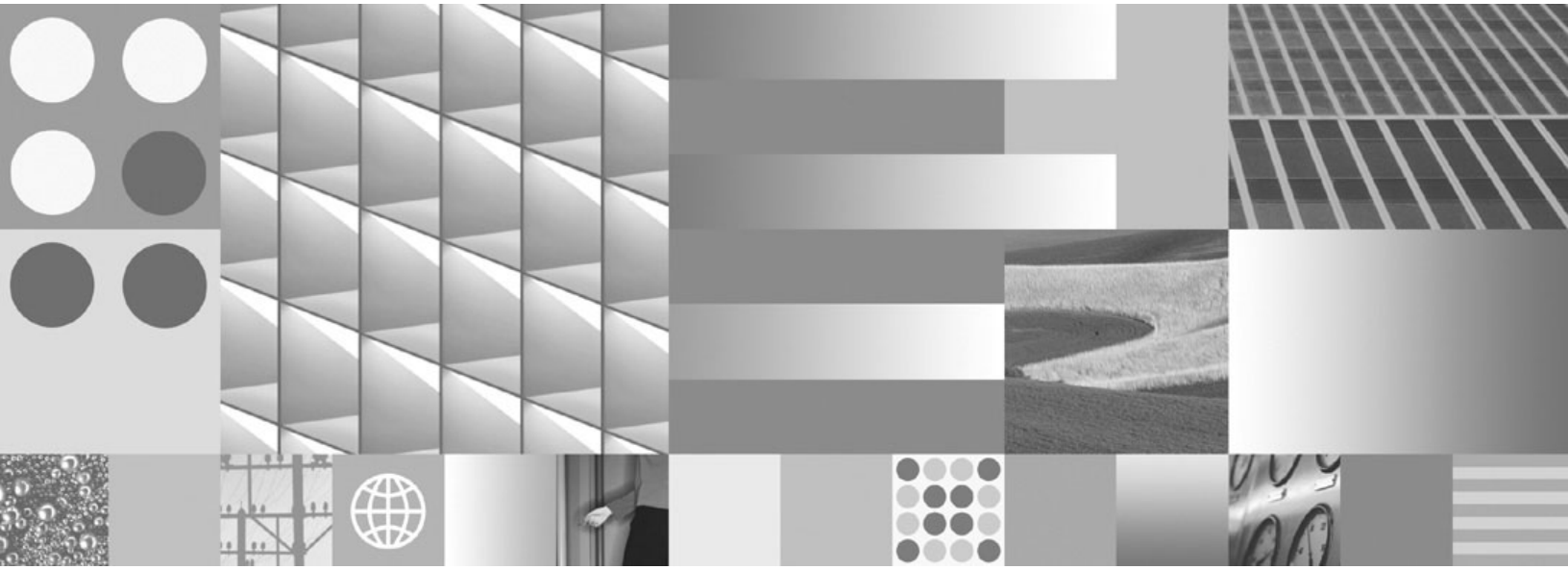
Desarrollo de rutinas definidas por el usuario (SQL y externas)

Actualizado en diciembre de 2010

SC11-3503-03

DB2 Versión 9.5
para Linux, UNIX y Windows

Versión 9 Release 5



Desarrollo de rutinas definidas por el usuario (SQL y externas)

Actualizado en diciembre de 2010

SC11-3503-03

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 389.

Nota de edición

Este manual es la traducción del original en inglés *DB2 Version 9.5 for Linux, UNIX, and Windows Version 9 Release 5 Developing User-defined Routines (SQL and External) Updated December, 2010* (SC23-5855-03).

Este documento contiene información propiedad de IBM. Se proporciona según un acuerdo de licencia y está protegido por la ley de la propiedad intelectual. La información contenida en esta publicación no incluye ninguna garantía de producto, por lo que ninguna declaración proporcionada en este manual deberá interpretarse como tal.

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.

© Copyright IBM Corporation 1993, 2010.

Contenido

Acerca de este manual vii
A quién va dirigido este manual vii
Cómo está estructurado este manual vii

Capítulo 1. Visión general de las rutinas 1
Rutinas 1
Beneficios del uso de rutinas 2
Tipos de rutinas 3
Rutinas definidas por el sistema y rutinas definidas por el usuario 5
Rutinas definidas por el sistema 5
Rutinas definidas por el usuario 6
Comparación de las rutinas definidas por el sistema y las rutinas definidas por el usuario . . 7
Determinar cuándo hay que utilizar rutinas definidas por el sistema o definidas por el usuario 8
Tipos de rutinas funcionales 9
Rutinas: Procedimientos 10
Rutinas: Funciones 11
Rutinas: Métodos 15
Comparación de los tipos funcionales de rutinas 16
Determinar qué tipo funcional de rutina hay que utilizar 20
Implementaciones de las rutinas 21
Implementación de rutinas incorporadas . . . 21
Implementación de una rutina con fuente . . . 22
Implementación de rutinas de SQL 22
Implementación de rutinas externas 22
Interfaces API y lenguajes de programación soportados para el desarrollo de rutinas externas . 23
Comparación de las interfaces API y los lenguajes de programación soportados para el desarrollo de rutinas externas 23
Comparación de implementaciones de rutinas . . 29
Determinar qué implementación de rutina hay que utilizar 32
Uso de las rutinas 32
Administración de bases de datos con rutinas definidas por el sistema 33
Ampliación del soporte a las funciones de SQL con las funciones definidas por el usuario . . . 33
Auditoría mediante funciones de tabla de SQL 34
Herramientas para desarrollar rutinas 37
Soporte para el desarrollo de rutinas de IBM Data Studio 37
Sentencias de SQL que pueden ejecutarse en rutinas 38
Niveles de acceso de SQL en rutinas 43
Determinar qué sentencias de SQL se pueden ejecutar en la rutina 44
Portabilidad de las rutinas 45
Interoperabilidad de las rutinas. 46
Rendimiento de las rutinas 46
Seguridad de las rutinas 54
Seguridad de las rutinas 56
Autorizaciones y enlace de rutinas que contienen SQL 57
Conflictos de datos cuando se leen o se graban procedimientos en tablas 60

Capítulo 2. Desarrollo de las rutinas . . 63

Capítulo 3. Visión general de las rutinas SQL 65
Rutinas de SQL 65
Herramientas para el desarrollo de rutinas SQL . . 66
Desarrollo de rutinas SQL en IBM Data Studio . 66
Lenguaje de procedimientos de SQL (SQL PL) . . . 66
SQL PL incorporado 67
SQL PL en procedimientos de SQL 69
PL de SQL incorporado y funciones de SQL, activadores y sentencias de SQL compuesto (dinámico) 69
Sentencias CREATE para rutinas de SQL . . . 71
Determinar cuándo deben utilizarse rutinas de SQL o rutinas externas 71
Determinar cuándo utilizar procedimientos o funciones de SQL 73
Determinar cuándo utilizar sentencias de SQL compuesto dinámico o procedimientos de SQL . . 74
Reescritura de procedimientos de SQL como funciones definidas por el usuario de SQL 75

Capítulo 4. Desarrollar procedimientos de SQL 79
Procedimientos de SQL 79
Características de los procedimientos de SQL . . . 79
Diseño de procedimientos de SQL 80
Componentes de los procedimientos de SQL . . 80
Estructura de los procedimientos de SQL . . . 81
Soporte a matrices en procedimientos SQL . . . 84
Parámetros en los procedimientos de SQL . . . 85
Marcadores de parámetros 86
Variables en los procedimientos de SQL (sentencias DECLARE, SET) 91
Soporte de XML y XQuery en procedimientos de SQL 92
Variables SQLCODE y SQLSTATE en procedimientos de SQL 93
Sentencias compuestas y ámbito de las variables en los procedimientos de SQL 94
Efecto de los compromisos y retrotracciones en los valores de variables y parámetros XML de los procedimientos de SQL 95
Cursores en los procedimientos de SQL 95
Cursores para expresiones XQuery en procedimientos de SQL 96
Elementos de lógica SQL PL en el cuerpo del procedimiento de SQL. 98

© Copyright IBM Corp. 1993, 2010

Sentencias relacionadas con variables en procedimientos de SQL 98
Sentencias condicionales en procedimientos de SQL 98
Sentencias en bucle en procedimientos de SQL 100
Sentencias de transferencia de control en procedimientos de SQL 103
Manejadores de condiciones en los procedimientos de SQL 107
Devolución de conjuntos de resultados desde procedimientos de SQL 107
Recepción de conjuntos de resultados de procedimiento en las rutinas de SQL 108
Creación de procedimientos de SQL 109
Creación de procedimientos de SQL desde la línea de mandatos 109
Personalización de las opciones de precompilación y vinculación para procedimientos de SQL 111
Mejora del rendimiento de los procedimientos de SQL 112

Capítulo 5. Visión general de las rutinas externas 119
Rutinas externas 119
Funciones de rutinas externas 119
Creación de rutinas externas 120
Gestión de bibliotecas y clases de rutinas externas 121
Despliegue de bibliotecas y clases de rutinas externas 122
Seguridad de archivos de bibliotecas o clases de rutinas externas 123
Resolución de bibliotecas y clases de rutinas externas 123
Modificaciones de archivos de bibliotecas y de clases de rutinas externas 124
Copia de seguridad y restauración de archivos de bibliotecas y de clases de rutinas externas . . 125
Gestión y rendimiento de bibliotecas de rutinas externas 125
Interfaces API y lenguajes de programación soportados para el desarrollo de rutinas externas . 125
Comparación de las interfaces API y los lenguajes de programación soportados para el desarrollo de rutinas externas 126
Soporte de 32 bits y 64 bits para rutinas externas 131
Rendimiento de las rutinas con bibliotecas de 32 bits en servidores de bases de datos de 64 bits . . 132
Estilos de parámetros de rutinas externas 133
Restricciones para rutinas externas 135

Capítulo 6. Desarrollo de rutinas externas 139
Características de las funciones y métodos externos 139
Funciones escalares externas 139
Modelo de proceso de los métodos y las funciones escalares externas 141
Funciones de tabla externas 142
Modelo de proceso de las funciones de tabla externas 142
Modelo de ejecución de las funciones para Java 144
Áreas reutilizables para funciones externas y métodos 145
Áreas reutilizables en sistemas operativos de 32 bits y 64 bits. 148
Soporte para el tipo de datos XML en las rutinas externas 149
Escritura de rutinas 150
Creación de rutinas externas 151
Rutinas CLR (Common Language Runtime) .NET 153
Soporte para el desarrollo de rutinas externas en lenguajes .NET CLR 154
Herramientas para desarrollar rutinas .NET CLR 155
Diseño de rutinas de CLR .NET 155
Creación de rutinas .NET CLR 164
Creación de código de rutinas .NET CLR . . . 167
Depuración de rutinas .NET CLR. 172
Ejemplos de rutinas .NET CLR 176
Rutinas C y C++ 216
Soporte para el desarrollo de rutinas externas en C 217
Soporte para el desarrollo de rutinas externas en C++ 218
Herramientas para el desarrollo de rutinas C y C++ 218
Diseño de rutinas C y C++ 218
Archivo de inclusión necesario para el desarrollo de rutinas C y C++ (sqludf.h) . . . 219
Parámetros en rutinas C y C++ 220
Tipos de datos de SQL soportados en rutinas C y C++ 233
Manejo de tipos de datos de SQL en rutinas C y C++ 236
Paso de argumentos a rutinas C, C++, OLE o COBOL 244
Variables gráficas del lenguaje principal en rutinas C y C++ 257
Decoración de tipos de C++ 257
Devolución de conjuntos de resultados procedentes de procedimientos C y C++ . . . 259
Creación de rutinas C y C++ 261
Creación del código de rutinas C y C++ 263
Creación de código de rutinas C y C++ mediante scripts bldrtn de ejemplo 263
Creación de códigos de rutinas C y C++ desde ventanas de mandatos de DB2. 269
Opciones de compilar y enlazar para rutinas C y C++ 271
Creación de procedimientos almacenados de SQL incorporado en C o C++ con archivos de configuración 282
Creación de funciones definidas por el usuario en C o C++ con archivos de configuración (AIX) 284
Reconstrucción de bibliotecas compartidas para rutinas DB2 284
Actualización del archivo de configuración del gestor de bases de datos. 285

Capítulo 7. Procedimientos COBOL 287
Soporte para el desarrollo de procedimientos externos en COBOL 289

Tipos de datos de SQL soportados en aplicaciones de SQL incorporado COBOL 289
Creación de rutinas COBOL 293
Opciones de compilar y enlazar para rutinas COBOL 293
Creación de rutinas IBM COBOL en AIX . . . 298
Creación de rutinas Micro Focus COBOL de UNIX 299
Creación de rutinas IBM COBOL en Windows 300
Creación de rutinas Micro Focus COBOL en Windows 301

Capítulo 8. Rutinas Java. 303
Software de desarrollo de rutinas Java soportado 304
Soporte de interfaces de programación de aplicaciones JDBC y SQLJ para rutinas Java . . . 304
Especificación de un SDK para el desarrollo de rutinas Java (UNIX) 305
Especificación de un controlador para rutinas Java 306
Herramientas para el desarrollo de rutinas Java (JDBC y SQLJ) 306
Diseño de rutinas Java 307
Tipos de datos de SQL soportados en rutinas Java 307
Contextos de conexión en rutinas SQLJ 309
Parámetros de las rutinas Java. 310
Estilo de parámetro en procedimientos JAVA 310
Funciones y métodos Java con parámetros de estilo JAVA 312
Rutinas DB2GENERAL 313
UDF DB2GENERAL 313
Tipos de datos de SQL soportados en rutinas DB2GENERAL 315
Clases de Java para rutinas DB2GENERAL . . 316
Clase DB2GENERAL de Java: COM.IBM.db2.app.StoredProc 317
Clase DB2GENERAL de Java: COM.IBM.db2.app.UDF 318
Clase DB2GENERAL de Java: COM.IBM.db2.app.Lob 320
Clase DB2GENERAL de Java: COM.IBM.db2.app.Blob 320
Clase DB2GENERAL de Java: COM.IBM.db2.app.Clob 321
Paso de parámetros de tipo ARRAY a rutinas Java 321
Devolución de conjuntos de resultados desde procedimientos JDBC. 322
Devolución de conjuntos de resultados desde procedimientos de SQLJ 323
Recepción de conjuntos de resultados de procedimiento en aplicaciones y rutinas JDBC . 324
Recepción de conjuntos de resultados de procedimiento en aplicaciones y rutinas SQLJ . 325
Restricciones de las rutinas Java 326
Modelo de ejecución de las funciones para Java 326
Creación de rutinas Java. 328
Creación de rutinas Java desde la línea de mandatos. 328
Creación del código de rutinas Java 331
Creación de rutinas JDBC 331
Creación de rutinas SQL. 332
Opciones de compilar y enlazar para rutinas Java (SQLJ) 333
Despliegue de archivos de clases de rutinas Java en servidores de base de datos DB2 334
Administración de los archivos JAR en el servidor de bases de datos 335
Actualización de clases de rutina Java 337
Ejemplos de rutinas Java (JDBC) 337
Ejemplo: tipo de datos de matriz en un procedimiento (JDBC) de Java 337
Ejemplo: Soporte de XML y XQuery en el procedimiento de Java (JDBC) 338

Capítulo 9. Diseño de rutinas de automatización de OLE 343
Creación de rutinas de automatización de OLE . . 343
Consideraciones sobre las instancias de objetos de rutinas OLE y el área reutilizable. 344
Tipos de datos de SQL soportados en la automatización de OLE 345
Rutinas de automatización de OLE en BASIC y C++ 346

Capítulo 10. Funciones de tabla de OLE DB definidas por el usuario . . . 349
Creación de una UDF para tablas OLE DB . . . 350
Nombres de conjunto de filas completamente calificados 351
Tipos de datos de SQL soportados en OLE DB . . 352

Capítulo 11. Invocación de rutinas 355
Autorizaciones y enlace de rutinas que contienen SQL 356
Vías de acceso y nombres de rutina 360
Invocaciones de rutinas anidadas. 361
Invocación de rutinas de 32 bits en un servidor de bases de datos de 64 bits 362
Consideraciones sobre la página de códigos de las rutinas 362
Referencias a procedimientos 363
Selección de procedimiento. 364
Llamada a procedimientos 364
Referencias a funciones 373
Selección de función 374
Tipos diferenciados como parámetros de UDF o método 375
Valores de LOB como parámetros de UDF. . . 376
Invocación de funciones escalares o métodos 377
Invocación de funciones de tabla definidas por el usuario 377

Apéndice A. Visión general de la información técnica de DB2 379
Biblioteca técnica de DB2 en copia impresa o en formato PDF 380
Pedido de manuales de DB2 en copia impresa . . 382
Visualización de la ayuda para estados de SQL desde el procesador de línea de mandatos. . . . 383

Acceso a diferentes versiones del Centro de información de DB2 384
Visualización de temas en su idioma preferido en el Centro de información de DB2 384
Actualización del Centro de información de DB2 instalado en el sistema o en el servidor de intranet . 385
Guías de aprendizaje de DB2 386
Información de resolución de problemas de DB2 387
Términos y condiciones 387

Apéndice B. Avisos 389

Índice 393

Acerca de este manual

Este manual contiene información sobre el desarrollo de rutinas definidas por el usuario cuando no está disponible una rutina definida por el sistema que ofrezca la funcionalidad necesaria.

A quién va dirigido este manual

Este manual está destinado a arquitectos de bases de datos, administradores de bases de datos y desarrolladores de aplicaciones a todos los niveles.

- Arquitectos de bases de datos interesados en saber cómo se pueden crear y utilizar objetos de rutinas para modularizar SQL y en la lógica relacionada que se puede reutilizar en varios contextos dentro de la arquitectura de bases de datos.
- Administradores de bases de datos interesados en saber cómo crear, gestionar, desplegar, proteger, solucionar y mejorar el rendimiento del sistema mediante el uso de rutinas definidas por el usuario dentro de un sistema de gestión de bases de datos.
- Desarrolladores de aplicaciones interesados en saber cómo y cuándo encapsular sentencias SQL y lógica de aplicaciones en rutinas para mejorar la modularidad de las aplicaciones y el rendimiento así como en el método paso a paso para diseñar, crear y generar rutinas definidas por el usuario. Los desarrolladores de aplicaciones deben tener experiencia en escribir sentencias SQL y experiencia de programación con uno de los lenguajes de programación soportados para el desarrollo de rutinas (C, C++, Java™, COBOL, C#, Visual Basic u otro lenguaje de programación soportado .NET CLR).

Cómo está estructurado este manual

Las principales áreas que se tratan en los capítulos de este manual son las siguientes:

Visiones generales

- El Capítulo 1, “Visión general de las rutinas”, presenta una visión general de las rutinas definidas por el sistema y por el usuario, los tipos funcionales, implementaciones, uso y herramientas para desarrollar rutinas.
- El Capítulo 3, “Visión general de las rutinas SQL”, ofrece información sobre las rutinas SQL y cómo determinar cuándo utilizar rutinas SQL, procedimientos o funciones.
- El Capítulo 5, “Visión general de las rutinas externas”, presenta información general sobre las rutinas externas.

Desarrollo de rutinas, procedimientos y funciones

- El Capítulo 2, “Desarrollo de rutinas”, describe a nivel general el procedimiento de desarrollo de rutinas.
- El Capítulo 4, “Desarrollo de procedimientos SQL”, describe las funciones, diseño y creación de procedimientos SQL.
- El Capítulo 6, “Desarrollo de rutinas externas”, incluye descripciones sobre el desarrollo de rutinas C, C++ y .NET CLR.

© Copyright IBM Corp. 1993, 2010

- El Capítulo 7, “Procedimientos COBOL”, presenta información sobre cómo desarrollar rutinas COBOL.
- El Capítulo 8, “Rutinas Java”, describe el desarrollo de rutinas Java.
- El Capítulo 9, “Diseño de rutinas de automatización OLE”. contiene información sobre el desarrollo de rutinas de automatización OLE (Object Linking and Embedding).
- El Capítulo 10, “Funciones de tablas definidas por el usuario OLE DB”, describe cómo crear funciones de tablas definidas por el usuario Object Linking and Embedding (OLE) DB.

Invocación de rutinas

- El Capítulo 11, “Invocación de rutinas”, describe cómo invocar la rutina una vez desarrollada.

Capítulo 1. Visión general de las rutinas

Las rutinas son un tipo de objeto de base de datos que puede utilizarse para encapsular lógica que puede invocarse como una subrutina de programación. Existen muchas aplicaciones de rutinas útiles en una base de datos o en la arquitectura de una de aplicación de base de datos. Las rutinas pueden utilizarse para mejorar el diseño general de la base de datos, el rendimiento de la base de datos y la seguridad de los datos, así como para implementar los mecanismos de auditoría básicos, etc.

Antes de decidir la implementación de una rutina, es importante que comprenda qué son las rutinas, cómo se implementan y cómo pueden utilizarse. Los temas sobre los conceptos siguientes lo ayudarán a comprender las rutinas para poder tomar decisiones informadas sobre cuándo y cómo utilizarlas en el entorno de bases de datos:

- "Beneficios del uso de rutinas" en la página 2
- "Tipos de rutinas" en la página 3
- "Herramientas para desarrollar rutinas" en la página 37
- "Uso de las rutinas" en la página 32
- "Niveles de acceso de SQL en rutinas" en la página 43
- "Creación de rutinas externas" en la página 120
- "Rendimiento de las rutinas" en la página 46
- "Seguridad de las rutinas" en la página 54

Una vez haya obtenido información sobre las rutinas, es posible que desee realizar una de las tareas siguientes:

- "Determinar cuándo hay que utilizar rutinas definidas por el sistema o definidas por el usuario" en la página 8
- "Determinar qué tipo funcional de rutina hay que utilizar" en la página 20
- "Determinar qué implementación de rutina hay que utilizar" en la página 32
- "Herramientas para desarrollar rutinas" en la página 37

Rutinas

Las rutinas son objetos de base de datos que pueden encapsular la lógica de programación y de base de datos que es posible invocar como una subrutina de programación desde distintas interfaces de SQL. Las rutinas pueden ser definidas por el sistema, lo que significa que se proporcionan con el producto, o definidas por el usuario, lo que significa que los usuarios pueden crearlas. Las rutinas pueden implementarse utilizando sentencias de SQL, un lenguaje de programación o una combinación de ambos. Distintos tipos de rutinas proporcionan interfaces diferentes que pueden utilizarse para ampliar las funciones de las sentencias de SQL, las aplicaciones cliente y algunos objetos de base de datos.

Para obtener una vista completa de los tipos de rutinas y de implementaciones soportadas por DB2, consulte el tema "Tipos de rutinas" en la página 3.

La multitud de características de las rutinas son el motivo por el que existen tantas aplicaciones útiles de las rutinas.

© Copyright IBM Corp. 1993, 2010

Beneficios del uso de rutinas

Se pueden obtener las ventajas siguientes mediante la utilización de rutinas:

Encapsular la lógica de aplicación que se puede invocar desde una interfaz de SQL En un entorno con distintas aplicaciones cliente que tengan requisitos comunes, el uso eficaz de las rutinas puede simplificar la reutilización, estandarización y mantenimiento del código. Si es necesario cambiar un aspecto determinado del comportamiento de una aplicación en un entorno en el que se utilizan rutinas, sólo deberá modificarse la rutina afectada que encapsule el comportamiento. Sin una rutina, será necesario modificar la lógica de aplicación de cada aplicación.

Permitir un acceso controlado a otros objetos de base de datos
Las rutinas pueden utilizarse para controlar el acceso a los objetos de base de datos. Puede que un usuario no tenga permiso para emitir generalmente una determinada sentencia de SQL, como por ejemplo CREATE TABLE; sin embargo, se le puede otorgar permiso para invocar rutinas que contengan una o varias implementaciones concretas de la sentencia, simplificando así la gestión de privilegios mediante la encapsulación de los privilegios.

Mejorar el rendimiento de las aplicaciones reduciendo el tráfico de la red
Cuando se ejecutan aplicaciones en un sistema cliente, cada sentencia de SQL se envía por separado desde el sistema cliente al sistema servidor de bases de datos que deba ejecutarse y cada conjunto de resultados se devuelve por separado. Esto puede derivar en niveles elevados de tráfico de red. Si es posible identificar un trabajo que requiera extensa interacción con la base de datos y escasa interacción con el usuario, resulta sensato instalar este trabajo en el servidor, para minimizar la cantidad de tráfico de red y permitir que el trabajo se realice en los servidores de bases de datos más potentes.

Permitir una ejecución de SQL más rápida y eficaz
Como las rutinas son objetos de base de datos, son más eficientes en la transmisión de datos y peticiones de SQL que las aplicaciones de cliente. Por tanto, las sentencias de SQL que se ejecuten en rutinas tendrán un rendimiento mejor que si se ejecutan en aplicaciones cliente. Las rutinas que se crean con la cláusula NOT FENCED se ejecutan en el mismo proceso que el gestor de bases de datos y, por tanto, pueden utilizar la memoria compartida para la comunicación, lo que puede dar como resultado que mejore el rendimiento de la aplicación.

Permitir la interoperatividad de la lógica implementada en distintos lenguajes de programación
Como los distintos programadores pueden implementar lenguajes de programación diferentes y como suele ser aconsejable reutilizar el código siempre que sea posible, las rutinas de DB2 proporcionan soporte a un alto grado de interoperatividad.

- Las aplicaciones cliente en un lenguaje de programación pueden invocar rutinas que estén implementadas en un lenguaje de programación distinto. Por ejemplo, las aplicaciones cliente en C pueden invocar rutinas CLR (Common Language Runtime) de .NET.
- Las rutinas pueden invocar otras rutinas con independencia del tipo de rutina o de la implementación de la rutina. Por ejemplo, un procedimiento de Java puede invocar una función escalar de SQL incorporado.

- Las rutinas creadas en un servidor de bases de datos de un sistema operativo pueden invocarse desde un cliente DB2 que se ejecute en un sistema operativo distinto.

Las ventanas que se han descrito antes son simplemente algunas de las múltiples ventajas asociadas con la utilización de las rutinas. La utilización de las rutinas puede resultar beneficiosa para una serie usuarios, entre los que se encuentran administradores de bases de datos, diseñadores de bases de datos y desarrolladores de aplicaciones de bases de datos. Por este motivo, existen muchas aplicaciones útiles de rutinas que es posible que desee explorar.

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

Tipos de rutinas

Existen muchos tipos distintos de rutinas. Las rutinas pueden agruparse de distintas formas pero principalmente se agrupan según si están definidas por el sistema o por el usuario, por su funcionalidad o por su implementación.

Las definiciones soportadas para las rutinas son las siguientes:
- "Rutinas definidas por el sistema" en la página 5
- "Rutinas definidas por el usuario" en la página 6

Los tipos funcionales soportados para las rutinas son los siguientes:
- "Rutinas: Procedimientos" en la página 10 (también denominados procedimientos almacenados)
- "Rutinas: Funciones" en la página 11
- "Rutinas: Métodos" en la página 15

Las implementaciones soportadas para las rutinas son las siguientes:
- "Implementación de rutinas incorporadas" en la página 21
- "Implementación de una rutina con fuente" en la página 22
- "Implementación de rutinas de SQL" en la página 22
- "Implementación de rutinas externas" en la página 22

El diagrama siguiente ilustra la jerarquía de clasificación de las rutinas. Todas las rutinas puede ser definidas por el sistema o definidas por el usuario. Los tipos funcionales de las rutinas aparecen en recuadros oscuros de color gris/azul y las implementaciones soportadas para las rutinas en recuadros claros de color gris/naranja. Se enfatiza la implementación de las rutinas incorporadas, ya que este tipo de implementación es exclusiva.

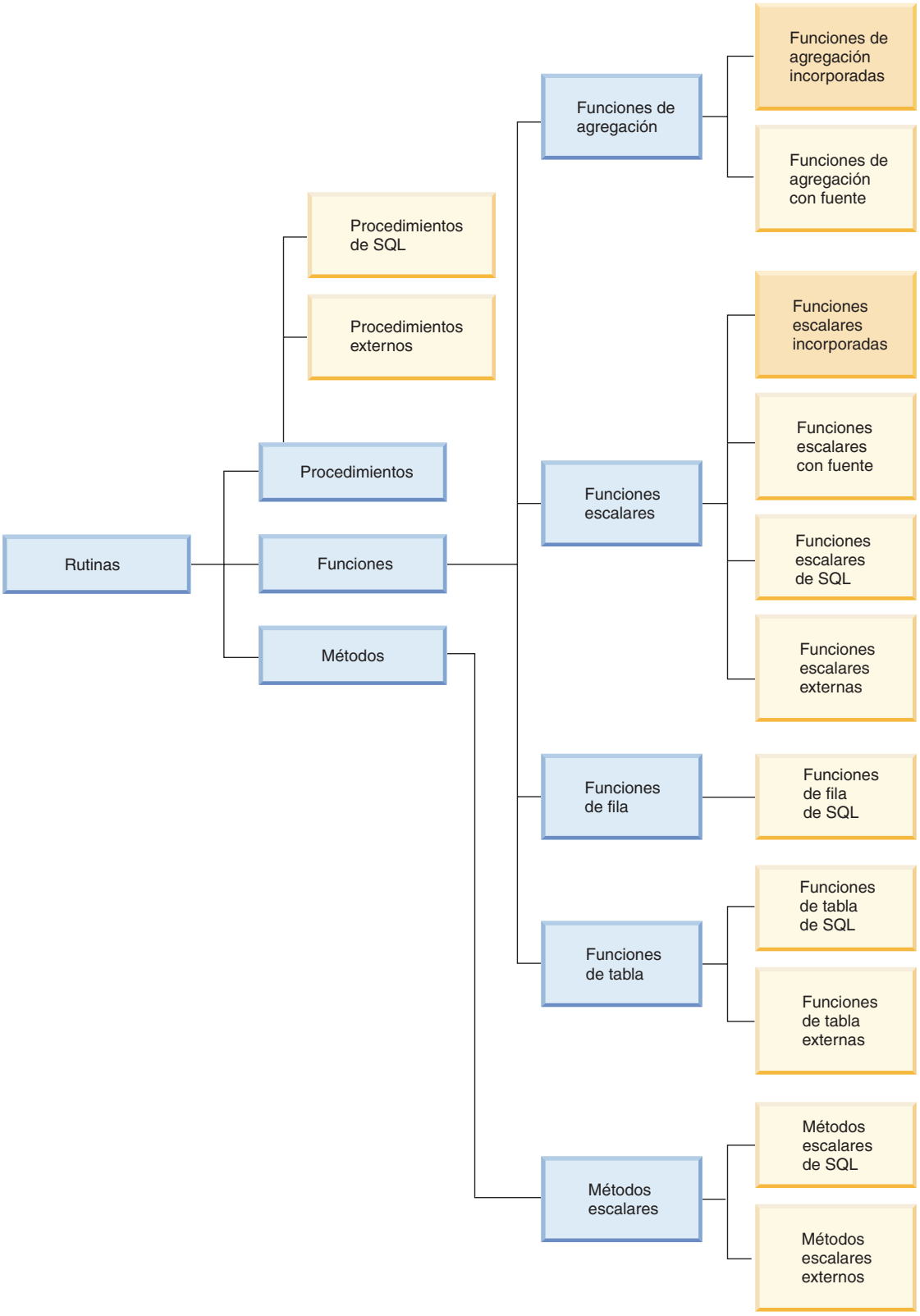


Figura 1. Clasificaciones de las rutinas

Los distintos tipos de rutinas proporciona amplio soporte para ampliar la funcionalidad del lenguaje SQL y desarrollar más aplicaciones de bases de datos modulares.

Rutinas definidas por el sistema y rutinas definidas por el usuario

Una de las formas más sencillas de clasificar las rutinas consiste es dividirlas en "Rutinas definidas por el sistema" y "Rutinas definidas por el usuario" en la página 6.

Las rutinas definidas por el sistema son rutinas que se proporcionan con el producto. Estas rutinas proporcionan un amplio soporte para la realización de tareas, un soporte que va de las funciones administrativas al informe de catálogos y sistemas de bases de datos. Están listas para su utilización de forma inmediata y no necesitan ningún paso de registro o prerrequisito para la configuración, aunque los usuarios requieren los privilegios necesarios para invocar estas rutinas.

Las rutinas definidas por el usuario son rutinas que los usuarios pueden crear ellos mismos. Las rutinas definidas por el usuario proporcionan una forma para que los usuarios amplíen el lenguaje SQL más allá del soporte disponible actualmente. Las rutinas definidas por el usuario pueden implementarse de distintas formas, entre las que se incluyen la utilización como fuente (la reutilización de la lógica) de las rutinas incorporadas, la utilización de sentencias de SQL exclusivamente o la utilización de SQL con otro lenguaje de programación.

Rutinas definidas por el sistema

Las rutinas definidas por el sistema son rutinas que se proporcionan con el producto. Estas rutinas proporcionan un amplio soporte a las rutinas para la realización de tareas, un soporte que va de las funciones administrativas al informe de catálogos y sistemas de bases de datos. Se caracterizan por estar listas para su utilización de forma inmediata y por no necesitar ningún paso de requisito previo para la configuración o el registro de las rutinas, aunque los usuarios necesitan privilegios para invocar estas rutinas. Éstas pueden incluir rutinas incorporadas y rutinas definidas por el sistema, que previamente se denominaban Rutinas administrativas de SQL.

Las rutinas incorporadas definidas por el sistema proporcionan soporte a los operadores estándar y soporte a las funciones de agregación y a las funciones escalares básicas. Las rutinas incorporadas constituyen la primera opción que debería utilizarse porque son de tipo firme y permiten obtener el mejor rendimiento. No cree rutinas externas que dupliquen el comportamiento de las rutinas incorporadas. Las rutinas externas no pueden tener un rendimiento tan bueno o ser tan seguras como las rutinas incorporadas.

En los esquemas SYSPROC, SYSFUN y SYSTOOLS se proporcionan otras rutinas definidas por el sistema para DB2. Estas rutinas son básicamente rutinas de SQL y rutinas externas que están definidas por el sistema y se proporcionan con el producto. Aunque estas rutinas adicionales se suministran con DB2, no son rutinas incorporadas. En lugar de ello, se implementan como rutinas definidas por el usuario preinstaladas. Habitualmente, estas rutinas encapsulan una función de programa de utilidad. Los siguientes son ejemplos de estas rutinas: *SNAP_GET_TAB_V91*, *HEALTH_DB_HI*, *SNAP_WRITE_FILE* y *REBIND_ROUTINE_PACKAGE*. Puede utilizar inmediatamente estas funciones y procedimientos siempre que tenga el esquema SYSPROC y el esquema SYSFUN en el registro especial CURRENT PATH. Resulta buena idea examinar el conjunto de rutinas definidas por el sistema proporcionadas para DB2 si piensa implementar una rutina externa para realizar funciones administrativas.

El procedimiento *ADMIN_CMD* puede resultarse especialmente útil, ya que proporciona una interfaz estándar para ejecutar muchos mandatos habituales de DB2 utilizando una interfaz de SQL.

Las rutinas definidas por el sistema agilizan y facilitan la implementación de consultas de SQL complejas y de aplicaciones de bases de datos potentes, ya que están listas para su utilización.

Rutinas definidas por el usuario

DB2 proporciona rutinas que capturan la funcionalidad de la mayoría de las funciones de conversión, de serie y aritméticas utilizadas normalmente. No obstante, a fin de que pueda encapsular su propia lógica, DB2 también permite al usuario crear sus propias rutinas. Estas rutinas se denominan rutinas definidas por el usuario. Puede crear sus propios "Rutinas: Procedimientos" en la página 10, "Rutinas: Funciones" en la página 11 y "Rutinas: Métodos" en la página 15 en cualquiera de los estilos de implementación soportados para el tipo de rutina. Generalmente, la expresión 'definido por el usuario' no se utiliza cuando se hace referencia a procedimientos y a métodos. Sin embargo, las funciones definidas por el usuario se suelen denominar UDF.

Creación de rutinas definidas por el usuario

Los procedimientos, funciones y métodos definidos por el usuario se crean en la base de datos ejecutando la sentencia CREATE adecuada para el tipo de rutina. Entre las sentencias de creación de rutinas se incluyen:

- "Sentencia CREATE PROCEDURE" en la publicación *Consulta de SQL*
- "Sentencia CREATE FUNCTION" en la publicación *Consulta de SQL*
- "Sentencia CREATE METHOD" en la publicación *Consulta de SQL*

Las cláusulas específicas de cada una de las sentencias CREATE definen las características de la rutina, tales como el nombre de la rutina, el número y tipo de argumentos de la rutina y detalles sobre la lógica de la misma. DB2 utiliza la información proporcionada por las cláusulas para identificar y ejecutar la rutina cuando ésta se invoca. Una vez ejecutada satisfactoriamente la sentencia CREATE para una rutina, ésta se crea en la base de datos. Las características de la rutina se almacenan en las vistas de catálogo de DB2 que los usuarios pueden consultar. La ejecución de la sentencia CREATE para crear una rutina también se conoce como definición de una rutina o registro de una rutina.

Las definiciones de rutinas definidas por el usuario se almacenan en el esquema de tabla del catálogo del sistema SYSTOOLS.

Implementación de lógica de rutina definida por el usuario

Existen tres estilos de implementación que se pueden utilizar para especificar la lógica de una rutina:

- "Implementación de una rutina con fuente" en la página 22: las rutinas definidas del usuario pueden tomar su *fuente* de la lógica de rutinas incorporadas existentes.
- "Implementación de rutinas de SQL" en la página 22: Las rutinas definidas por el usuario pueden implementarse utilizando solamente sentencias de SQL.
- "Implementación de rutinas externas" en la página 22: las rutinas definidas por el usuario pueden implementarse utilizando uno de los conjuntos de lenguajes de programación soportados.

Cuando se crean rutinas en un lenguaje de programación que no es SQL, la biblioteca o clase creada a partir del código se asocia con la definición de rutina por el valor especificado en la cláusula EXTERNAL NAME. Cuando se invoca la rutina, se ejecuta la biblioteca o clase asociada con la rutina.

Las rutinas definidas por el usuario pueden incluir bastantes sentencias de SQL pero no todas las sentencias de SQL.

Las rutinas definidas por el usuario son de tipo firme pero los desarrolladores de las rutinas deben desarrollar o mejorar los mecanismos de gestión de tipos y de gestión de errores.

Al realizarse la migración de base de datos podría ser necesario verificar o actualizar implementaciones de rutinas.

En general, las rutinas definidas por el usuario tienen un rendimiento satisfactorio, pero no tan bueno como el de las rutinas definidas por el sistema.

Las rutinas definidas por el usuario pueden invocar las rutinas definidas por el sistema y a otras rutinas definidas por el usuario implementadas en cualquier de los formatos soportados. Esta flexibilidad permite a los usuarios tener esencialmente la libertad de crear una biblioteca completa de los módulos de rutinas que pueden reutilizarse.

En general, las rutinas definidas por el usuario proporcionan una forma de ampliar el lenguaje SQL y de modularizar la lógica que se reutilizará en múltiples consultas o aplicaciones de bases de datos en las que no existan rutinas definidas por el sistema.

Comparación de las rutinas definidas por el sistema y las rutinas definidas por el usuario

Comprender las diferencias entre las rutinas definidas por el sistema y las definidas por el usuario puede ayudarlo a determinar si realmente necesita crear sus propias rutinas o reutilizar rutinas existentes. La posibilidad de determinar cuándo reutilizar rutinas existentes y cuándo desarrollar sus propias rutinas puede ahorrarle tiempo y esfuerzo, así como garantizarle la maximización del rendimiento de las rutinas.

Las rutinas definidas por el sistema y las rutinas definidas por el usuario difieren en una serie de aspectos. Estas diferencias se resumen en la tabla siguiente:

Tabla 1. Comparación de las rutinas definidas por el sistema y definidas por el usuario

Característica	Rutinas definidas por el sistema	Rutinas definidas por el usuario
Soporte de características	Funcionalidad amplia de operadores numéricos, manipulación de series y administración disponible para su utilización de inmediato. Para utilizar estas rutinas, sencillamente invoque las rutinas desde interfaces soportadas.	Aunque no todas las sentencias de SQL están soportadas en las rutinas definidas por el usuario, la gran mayoría sí lo están. También es posible derivar llamadas a rutinas definidas por el sistema en rutinas definidas por el usuario si desea ampliar la funcionalidad de las rutinas definidas por el sistema. Las rutinas definidas por el usuario proporcionan una oportunidad ilimitada para la implementación de la lógica de la rutina. Para utilizar estas rutinas, debe desarrollarlas primero y a continuación invocarlas desde interfaces soportadas.
Mantenimiento	No se requiere ningún mantenimiento.	Las rutinas externas requieren la gestión de las bibliotecas de rutinas externas asociadas.
Migración	Ningún efecto sobre las migraciones o efecto muy limitado.	Es posible que para migraciones de release a release deba verificar las rutinas.
Rendimiento	Rendimiento mejor que el de las rutinas definidas por el usuario equivalentes.	Generalmente no funcionan tan bien como las rutinas definidas por el sistema equivalentes.
Estabilidad	Soporte de tipos firmes y gestión de errores.	El desarrollador de la rutina debe programar el soporte de tipos y la gestión de errores.

Siempre que sea posible, debería optar por la utilización de las rutinas definidas por el sistema. Éstas se proporcionan para facilitar el desarrollo de aplicaciones y la formulación de sentencias de SQL y están optimizadas para conseguir un buen rendimiento. Las rutinas definidas por el usuario le proporcionan la flexibilidad para crear sus propias rutinas en caso de que ninguna de las rutinas definidas por el sistema realice la lógica empresarial específica que desea implementar.

Determinar cuándo hay que utilizar rutinas definidas por el sistema o definidas por el usuario

Las rutinas definidas por el sistema proporcionan funciones encapsuladas, listas para usar y que ahorran tiempo, mientras que las rutinas definidas por el usuario proporcionan flexibilidad, que le permite definir sus propias rutinas cuando no hay rutinas definidas por el sistema que contengan adecuadamente la función que se necesita.

Para determinar si hay que utilizar una rutina definida por el sistema o una rutina definida por el usuario, haga lo siguiente:

1. Determine qué función necesita que esté encapsulada en la rutina.
2. Consulte la lista de rutinas definidas por el sistema disponibles para ver si hay alguna que coincida con uno de los requisitos o con todos ellos.

- Si hay una rutina definida por el sistema que coincida con algunos requisitos, pero no con todos ellos:
  - Determine si la función que falta se puede añadir sencillamente a la aplicación. Si es así, utilice la rutina definida por el sistema y modifique la aplicación para que cubra la función que falta. Si la función que falta no resulta fácil de añadir a la aplicación o si la función que falta se debe repetir en numerosos lugares, plantéese la posibilidad de crear una rutina definida por el usuario que contenga la función que falta y que invoque la rutina definida por el sistema.
  - Si piensa que los requisitos de la rutina irán cambiando y que tal vez tenga que modificar con frecuencia la definición de la rutina, plantéese la posibilidad de usar una rutina definida por el usuario en lugar de una rutina definida por el sistema.
  - Determine si hay parámetros adicionales que le interese pasar a la rutina o viceversa. Si los hay, puede ser conveniente crear una rutina definida por el usuario que encapsule una invocación a la rutina definida por el sistema.
- Si no hay ninguna rutina definida por el sistema que capture adecuadamente la función que desea encapsular, cree una rutina definida por el usuario.

Para ahorrar tiempo y trabajo, es conveniente que utilice rutinas definidas por el sistema siempre que sea posible. Habrá veces en que la función que necesita no está disponible en una rutina definida por el sistema. En esos casos, debe crear una rutina definida por el usuario. Otras veces puede ser posible incluir una llamada a una rutina definida por el sistema desde una rutina definida por el usuario que cubra la función adicional que se necesita.

Tipos de rutinas funcionales

Existen distintos tipos funcionales de rutinas. Cada uno de los tipos funcionales proporciona soporte para invocar rutinas de interfaces diferentes para distintos fines. Cada tipo funcional de rutina proporciona un conjunto de características y de soporte a SQL distinto.

- Los "Rutinas: Procedimientos" en la página 10, también denominados procedimientos almacenados, sirven de extensiones de subrutina a las aplicaciones cliente, rutinas, activadores y sentencias dinámicas compuestas. Los procedimientos se invocan ejecutando la sentencia CALL con una referencia a un procedimiento. Los procedimientos pueden tener parámetros de entrada, de salida y de entrada-salida, ejecutar una amplia gama de sentencias de SQL y devolver varios conjuntos de resultados al que llama.
- Las "Rutinas: Funciones" en la página 11 son relaciones entre conjuntos de valores de datos de entrada y un conjunto de valores de resultado. Las funciones le permiten ampliar y personalizar el SQL. Las funciones se invocan desde elementos de sentencias de SQL tales como una lista de selección, una expresión o una cláusula FROM. Existen cuatro tipos de funciones: funciones de agregación, funciones escalares, funciones de fila y funciones de tabla.
- Los "Rutinas: Métodos" en la página 15 permiten acceder a los atributos de tipos definidos por el usuario así como definir comportamientos adicionales para los tipos definidos por el usuario. Un tipo estructurado es un tipo de datos definido por el usuario que contiene uno o más atributos con nombre, cada uno de los

cuales tiene un tipo de datos. Los atributos son propiedades que describen una instancia de un tipo. Por ejemplo, una figura geométrica puede tener atributos tales como su lista de coordenadas cartesianas. Un método para un tipo estructurado se implementa generalmente como una operación sobre los atributos del tipo estructurado. Para una figura geométrica, un método puede consistir en calcular el volumen de ésta.

Para obtener detalles concretos sobre cada uno de los tipos de rutinas funcionales, consulte los temas para cada tipo de rutina.

Rutinas: Procedimientos

Los procedimientos, también denominados procedimientos almacenados, son objetos de base de datos creados ejecutando la sentencia CREATE PROCEDURE. Los procedimientos pueden encapsular la lógica y las sentencias de SQL y pueden servir de ampliaciones de subrutinas para las aplicaciones cliente, rutinas, activadores y sentencias dinámicas compuestas. Los procedimientos se invocan ejecutando la sentencia CALL con una referencia a un procedimiento. Los procedimientos pueden admitir parámetros de entrada, de salida y de entrada-salida, ejecutar una amplia gama de sentencias de SQL y devolver varios conjuntos de resultados a quien los utiliza.

Características

- Permiten la encapsulación de elementos lógicos y de sentencias de SQL que formulan un módulo de subrutina concreto
- Se pueden llamar desde aplicaciones cliente, otras rutinas, activadores y sentencias dinámicas compuestas; desde cualquier lugar en que pueda ejecutarse una sentencia CALL.
- Devuelven varios conjuntos de resultados
- Soportan la ejecución de un amplio conjunto de sentencias de SQL, incluidas las sentencias de SQL que lean o modifiquen datos de tabla en bases de datos de una sola partición y de varias
- Soporte paramétrico a los parámetros de entrada, salida y entrada-salida
- Se proporciona soporte a las llamadas anidadas a los procedimientos y a las invocaciones de funciones
- Se proporciona soporte a las llamadas recursivas a los procedimientos
- Se proporciona soporte a los puntos de salvaguarda y control de transacciones en los procesos

Limitaciones

- Los procedimientos no pueden invocarse desde una sentencia de SQL a excepción de la sentencia CALL. Como alternativa, se pueden utilizar las funciones para expresar la lógica que transforma los valores de columna.
- Otra sentencia de SQL no puede utilizar directamente los valores de los parámetros de salida y los conjuntos de resultados de las llamadas a procedimientos. Debe utilizarse la lógica de la aplicación para asignarlos a variables que puedan utilizarse en sentencias de SQL posteriores.
- Los procedimientos no pueden conservar el estado entre las invocaciones.
  - Consulte el tema "Restricciones de los procedimientos"

Usos frecuentes

- Estandarización de la lógica de la aplicación

- Si varias aplicaciones deben realizar el acceso o modificación de la base de datos de manera similar, un procedimiento puede proporcionar una única interfaz para la lógica. El procedimiento está entonces disponible para volverse a utilizar. En caso de que la interfaz deba modificarse para acomodar un cambio en la lógica empresarial, sólo se deberá modificar el procedimiento.

- Aislamiento de operaciones de base de datos de lógica que no sea de base de datos en las aplicaciones
  - Los procedimientos facilitan la implementación de las subrutinas que encapsulan los accesos a bases de datos y la lógica asociados con una tarea concreta que pueden reutilizarse en varias instancias. Por ejemplo, una aplicación de gestión de empleados puede encapsular las operaciones de base de datos específicas para la tarea de contratar un empleado. Un procedimiento de este tipo podría insertar la información sobre los empleados en varias tablas, calcular el sueldo semanal de los empleados basándose en un parámetro de entrada, y devolver el valor del sueldo semanal como un parámetro de salida. Otro procedimiento podría realizar un análisis estadístico de los datos de una tabla y devolver los conjuntos de resultados que contuvieran los resultados del análisis.
- Simplificación de la gestión de privilegios para un grupo de sentencias de SQL
  - Al permitir la encapsulación de una agrupación de varias sentencias de SQL en un objeto de base de datos con nombre, los procedimientos hacen posible que los administradores de bases de datos gestionen menos privilegios. En lugar de tener que otorgar los privilegios necesarios para ejecutar cada una de las sentencias de SQL de la rutina, sólo deben gestionar los privilegios para invocar la rutina.

Implementaciones soportadas

- Hay procedimientos definidos por el sistema listos para su utilización y los usuarios pueden crear procedimientos definidos por el usuario. Para los procedimientos, se soportan las siguientes implementaciones definidas por el usuario:
  - Implementación de SQL
  - Implementación externa
    - Consulte el tema "Lenguajes de programación para la implementación de rutinas externas soportados".

Rutinas: Funciones

Las funciones son relaciones entre conjuntos de valores de datos de entrada y un conjunto de valores de resultado. Le permiten ampliar y personalizar el SQL. Las funciones se invocan desde elementos de sentencias de SQL tales como una lista de selección o una cláusula FROM. Existen cuatro tipos de funciones:

- “Funciones de agregación” en *Consulta de SQL, Volumen 1*
- Funciones escalares
- Funciones de fila
- Funciones de tabla

Funciones de agregación

También denominada función de columna, este tipo de función devuelve un valor escalar que es el resultado de un cálculo con un conjunto de valores de entrada semejantes. Los valores de entrada similares se pueden

especificar, por ejemplo, mediante una columna dentro de una tabla o mediante tuplas en una cláusula VALUES. Este conjunto de valores se conoce como conjunto de argumentos. Por ejemplo, la consulta siguiente busca la cantidad total de cerrojos de los que se dispone o los que están pedidos utilizando la función de agregación SUM:

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

Funciones escalares

Una función escalar es una función que, para cada conjunto de uno o más parámetros escalares, devuelve un solo valor escalar. Entre las funciones escalares se incluyen la función LENGTH y la función SUBSTR. También es posible crear funciones escalares que realicen cálculos matemáticos complejos en los parámetros de entrada de las funciones. Se puede hacer referencia a las funciones escalares en cualquier parte donde una expresión sea válida dentro de una sentencia de SQL, como, por ejemplo, en una lista de selección o en una cláusula FROM. El ejemplo siguiente muestra una consulta que hace referencia a la función escalar incorporada LENGTH:

```
SELECT lastname, LENGTH(lastname)
FROM employee
```

Funciones de fila

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

Funciones de tabla

Funciones de tabla son las que, para un grupo de conjuntos de uno o más parámetros, devuelven una tabla a la sentencia de SQL que hace referencia a las mismas. Sólo se puede hacer referencia a las funciones de tabla en la cláusula FROM de una sentencia SELECT. La tabla devuelta por una función de tabla puede participar en uniones, operaciones de agrupación, operaciones como UNION y cualquier operación que se pueda aplicar a una vista de sólo lectura. El ejemplo siguiente hace una demostración de una función de tabla SQL que actualiza una tabla de inventario y devuelve el conjunto de resultados en la tabla de inventario actualizada.

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

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

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

Las funciones proporcionan soporte a las características siguientes:

- Las funciones están soportadas en todos los productos de base de datos de la marca DB2 que incluyen, entre otros, DB2, DB2 para z/OS y DB2 Database para System i

- Soporte moderado para la ejecución de sentencias de SQL
- Soporte de parámetros para los parámetros de entrada y los valores de retorno de funciones escalares y de agregación
- Compilación eficaz de la lógica de las funciones en las consultas que hacen referencia a funciones
- Las funciones externas proporcionan soporte para almacenar valores intermedios entre las subinvocaciones a funciones individuales para cada fila o valor

Hay funciones definidas por el sistema listas para su utilización y los usuarios pueden crear funciones definidas por el usuario. Las funciones pueden implementarse como funciones de SQL o como funciones externas. La implementación de las funciones de SQL resulta sencilla. Los métodos externos proporcionan soporte para la implementación flexible de la lógica, permiten que el usuario desarrolle la lógica del método en el lenguaje de programación que prefiera, y proporcionan capacidad para almacenar valores intermedios.

Rutinas: Funciones escalares

Una función escalar es una función que, para cada conjunto de uno o más parámetros escalares, devuelve un solo valor escalar. Entre las funciones escalares se incluyen la función LENGTH y la función SUBSTR. También es posible crear funciones escalares que realicen cálculos matemáticos complejos en los parámetros de entrada de las funciones. Se puede hacer referencia a las funciones escalares desde cualquier parte donde una expresión sea válida dentro de una sentencia de SQL, como, por ejemplo, en una lista de selección o en una cláusula FROM.

Características

- El rendimiento de las funciones escalares incorporadas es bueno.
- Las funciones escalares incorporadas tienen tipos firmes.
- Se puede hacer referencia a ellas con sentencias de SQL en todos aquellos lugares en que se soportan expresiones.
- La lógica se ejecuta en el servidor como parte de la sentencia de SQL que hace referencia a la misma.
- La sentencia que hace referencia a la función puede utilizar la salida de una UDF escalar directamente.
- Cuando se utiliza en predicados, el uso de la UDF escalar puede mejorar el rendimiento general de la consulta. Cuando una función escalar se aplica a un conjunto de filas candidatas del servidor, ésta puede actuar como un filtro, limitando así el número de filas que deben devolverse al cliente.
- Para las funciones escalares externas definidas por el usuario, se puede mantener el estado entre las invocaciones iterativas de la función empleando un área reutilizable.

Limitaciones

- Por su diseño, sólo devuelven un solo valor escalar.
- La gestión de transacciones no está soportada en las funciones escalares. Las confirmaciones y retrotracciones no pueden ejecutarse en los cuerpos de las funciones escalares.
- No se pueden devolver conjuntos de resultados desde funciones escalares.
- En una base de datos de una sola partición, las UDF escalares externas definidas por el usuario pueden incluir sentencias de SQL. Estas sentencias pueden leer datos de tablas, pero no pueden modificarlos.

- En un entorno de bases de datos de varias particiones, las UDF escalares definidas por el usuario no pueden incluir sentencias de SQL.
- Consulte: Restricciones de las funciones escalares.

Usos frecuentes

- Manipular series en sentencias de SQL.
- Realizar operaciones matemáticas básicas en sentencias de SQL.
- Es posible crear funciones escalares definidas por el usuario para ampliar el conjunto existente de funciones escalares incorporadas. Por ejemplo, puede crear una función matemática compleja reutilizando las funciones escalares incorporadas existentes junto con otra lógica.

Implementaciones soportadas

- Implementación con fuente
- Implementación externa
  - Consulte el tema, "Comparación de las interfaces API y los lenguajes de programación soportados para el desarrollo de rutinas externas" en la página 23.

Rutinas: Funciones de fila

Una función de fila es una función que sólo puede utilizarse con tipos de estructuras definidos por el usuario que, para cada conjunto de uno o más parámetros escalares, devuelve una sola fila. Las funciones de fila sólo se pueden utilizar como atributos de correlación de funciones de transformación de un tipo estructurado en valores de tipos de datos incorporados en una fila. Las funciones de fila no pueden utilizarse de forma autónoma ni en sentencias de SQL fuera del contexto de los tipos de datos abstractos.

Características

- Le permiten correlacionar atributos de tipos estructurados con una fila de valores de tipos de datos incorporados.

Limitaciones

- No pueden utilizarse de forma autónoma ni en sentencias de SQL fuera del contexto de los tipos estructurados definidos por el usuario.
- Consulte el tema "Restricciones de las funciones de fila"

Usos frecuentes

Para hacer accesibles los atributos de tipos estructurados en consultas u operaciones. Por ejemplo, supongamos que un tipo de datos estructurado definido por el usuario denominado 'gestor' se extiende sobre otro tipo estructurado, de persona, y que éste tiene una combinación de atributos de persona y de atributos específicos del gestor. Si desea hacer referencia a estos valores en una consulta, podría crear una función de fila para convertir los valores de los atributos en una fila de valores a los que pudiera hacerse referencia.

Implementaciones soportadas

- Implementación de SQL

Rutinas: Funciones de tabla

Las funciones de tabla son funciones que para un grupo de conjuntos de uno o más parámetros devuelven una tabla a la sentencia de SQL que hace referencia a las mismas. Sólo se puede hacer referencia a las funciones de tabla en la cláusula FROM de una sentencia SELECT. La tabla devuelta por una función de tabla puede

participar en uniones, operaciones de agrupación, operaciones como UNION y cualquier operación que se pueda aplicar a una vista de sólo lectura.

Características

- Devuelven un conjunto de valores de datos para su proceso.
- Se puede hacer referencia a ellas como parte de una consulta SQL.
- Pueden efectuar llamadas al sistema operativo, leer datos de archivos o incluso acceder a datos a través de una red en una base de datos de una sola partición.
- La sentencia de SQL que haga referencia a la función de tabla puede acceder directamente a los resultados de las invocaciones de la función de tabla.
- Las funciones de tabla SQL pueden encapsular sentencias de SQL que modifiquen datos de tablas SQL. Las funciones de tabla externas no pueden encapsular sentencias de SQL.
- Para una sola referencia de función de tabla, una función de tabla se puede invocar de forma iterativa varias veces y mantener el estado entre estas invocaciones utilizando un área reutilizable.

Limitaciones

- La gestión de transacciones no está soportada en las funciones de tabla definidas por el usuario. Las confirmaciones y retrotracciones no pueden ejecutarse en UDF de tabla.
- No se pueden devolver conjuntos de resultados desde funciones de tabla.
- No están diseñadas para invocaciones simples.
- Sólo se puede hacer referencia a las mismas en la cláusula FROM de una consulta.
- Las funciones de tabla definidas por el usuario pueden leer datos SQL, pero no pueden modificar datos SQL. De forma alternativa, es posible utilizar funciones de tabla SQL de modo que incluyan sentencias que modifiquen los datos SQL.
- Consulte el tema "Restricciones de las funciones de tabla"

Usos frecuentes

- Encapsular una subconsulta compleja, pero utilizada habitualmente.
- Proporcionar una interfaz tabular para datos no relacionales. Por ejemplo, una función de tabla externa definida por el usuario puede leer una hoja de cálculo y generar una tabla de valores que puede insertarse directamente en una tabla o accederse a la misma de forma directa e inmediata dentro de una consulta.

Implementaciones soportadas

- Implementación de SQL
- Implementación externa

Rutinas: Métodos

Los métodos permiten acceder a los atributos de tipo estructurado así como definir comportamientos adicionales para los tipos estructurados. Un tipo estructurado es un tipo de datos definido por el usuario que contiene uno o más atributos con nombre, cada uno de los cuales tiene un tipo de datos. Los atributos son propiedades que describen una instancia de un tipo. Por ejemplo, una figura geométrica puede tener atributos tales como su lista de coordenadas cartesianas.

Los métodos suelen implementarse para un tipo estructurado a fin de representar operaciones sobre los atributos del tipo estructurado. Para una figura geométrica, un método puede consistir en calcular el volumen de ésta. Los métodos comparten todas las características de las funciones escalares.

Características

- Posibilidad de acceder a los atributos de tipos estructurados
- Posibilidad de establecer atributos de tipos estructurados
- Posibilidad de crear operaciones sobre atributos de tipos estructurados y de devolver un valor de función
- Sensibilidad al tipo dinámico del tipo sujeto

Limitaciones

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

Usos frecuentes

- Crear operaciones sobre tipos estructurados
- Encapsular el tipo estructurado

Implementaciones soportadas

No hay ningún método definido por el sistema. Los usuarios pueden crear métodos definidos por el usuario para tipos estructurados definidos por el usuario existentes. Los métodos pueden implementarse utilizando una de las implementaciones siguientes:

- "Implementación de rutinas de SQL" en la página 22
- "Implementación de rutinas externas" en la página 22: C, C++, Java, C# (utilizando API OLE), Visual Basic (utilizando API OLE)

La implementación de los métodos de SQL resulta sencilla pero suelen estar diseñados en conjunto con el diseño de un tipo estructurado. Los métodos externos proporcionan mayor soporte a la implementación flexible de la lógica y permiten que el usuario desarrolle la lógica del método en el lenguaje de programación que prefiera.

Comparación de los tipos funcionales de rutinas

El hecho de comprender las diferencias entre procedimientos, funciones y métodos puede ayudarle a determinar qué tipo funcional hay que implementar al construir rutinas propias y determinar dónde y cómo se puede hacer referencia a las rutinas existentes. Esto puede ahorrarle tiempo y esfuerzo así como garantizarle la maximización de la funcionalidad y el rendimiento de las rutinas.

"Rutinas: Procedimientos" en la página 10, "Rutinas: Funciones" en la página 11 y "Rutinas: Métodos" en la página 15 difieren en una serie de aspectos. Estas diferencias se resaltan en la tabla siguiente:

Tabla 2. Comparación de los tipos funcionales de rutinas

Características	Procedimientos	Funciones	Métodos
Características funcionales exclusivas y aplicaciones útiles	• Permiten la encapsulación de la lógica y las sentencias de SQL. • Sirven de extensiones de subrutina a las aplicaciones cliente, rutinas, activadores y sentencias dinámicas compuestas. • Los procedimientos se invocan ejecutando la sentencia CALL con una referencia a un procedimiento. • Se proporciona soporte a las llamadas anidadas a procedimientos • Se proporciona soporte a las llamadas recursivas a procedimientos • Soporte paramétrico a los parámetros de entrada, salida y entrada-salida • Amplio soporte a la ejecución de sentencias de SQL • Pueden devolver uno o más conjuntos de resultados • Puntos de salvaguarda y control de transacciones	• Permiten la encapsulación de la lógica y las sentencias de SQL. • Las funciones son relaciones entre conjuntos de valores de datos de entrada y un conjunto de valores de resultado. • Las funciones le permiten ampliar y personalizar el SQL. • Las funciones se invocan desde elementos de sentencias de SQL tales como una lista de selección o una cláusula FROM. • Soporte moderado a la ejecución de sentencias de SQL. • Soporte de parámetros para los parámetros de entrada y los valores de retorno de funciones escalares y de agregación • Las funciones externas proporcionan soporte para almacenar valores intermedios entre las subinvocaciones a funciones individuales para cada fila o valor utilizando un área reutilizable. • Compilación eficaz de la lógica de las funciones en las consultas que hacen referencia a funciones.	• Permiten la encapsulación de la lógica y las sentencias de SQL. • Los métodos permiten acceder a los atributos de tipos estructurados así como definir comportamientos adicionales para los tipos estructurados. • Posibilidad de acceder a los atributos de tipos estructurados. • Posibilidad de establecer atributos de tipos estructurados. • Posibilidad de crear operaciones sobre atributos de tipos estructurados y de devolver un valor de función.

Tabla 2. Comparación de los tipos funcionales de rutinas (continuación)

Características	Procedimientos	Funciones	Métodos
Subtipos funcionales de rutinas	• No aplicable	• Funciones escalares • Funciones de agregación • Funciones de fila • Funciones de tabla	• No aplicable
Interfaz de invocación	• La invocación se realiza ejecutando la sentencia CALL con una referencia al procedimiento. • La invocación de procedimientos está soportada en aquellos lugares en que se proporcione soporte a la sentencia CALL.	• La invocación se realiza, entre otras ubicaciones, en una sentencia de SQL contenida en una lista de selección de columnas, una expresión o una cláusula FROM de una sentencia de selección.	• La invocación se realiza en una sentencia de SQL que haga referencia al tipo estructurado asociado con el método.
¿Existe alguna rutina definida por el usuario de este tipo?	• Sí, muchas. • En la publicación de consulta SQL se incluye una lista de los procedimientos definidos por el sistema.	• Sí, muchas. • En la publicación de consulta SQL se incluye una lista de las funciones definidas por el sistema.	• No
Implementaciones de rutinas definidas por el usuario soportadas	• SQL • Externas – C/C++ (con llamadas API CLI o SQL incorporadas) – COBOL – Java (JDBC) – Java (SQLJ) – CLR .NET – OLE: Visual Basic, Visual C++	• SQL • Externas – C/C++ – Java (JDBC) – Java (SQLJ) – CLR .NET – OLE DB: Visual Basic, Visual C++ (solo para funciones de tabla)	• SQL • Externas – C – C++
Soporte a llamadas anidadas	Sí	No. Sin embargo, las funciones se invocan repetidamente para cada uno de los valores del conjunto de entrada y los valores intermedios pueden almacenarse utilizando un área reutilizable.	No

Tabla 2. Comparación de los tipos funcionales de rutinas (continuación)

Características	Procedimientos	Funciones	Métodos
Rendimiento	El rendimiento es bueno si la lógica de la rutina es eficaz y se adoptan las mejores prácticas.	• El rendimiento es bueno si la lógica de la rutina es eficaz y se adoptan las mejores prácticas. • Puede tener un rendimiento mejor que un procedimiento equivalente desde el punto de vista lógico si la lógica solo consulta los datos, pero no los modifica.	Buen rendimiento
Portabilidad	• Portabilidad elevada • Portabilidad especialmente elevada si se utiliza la implementación de SQL. • Se proporciona soporte a las rutinas externas de 32 bits y 64 bits en distintos lenguajes de programación	• Portabilidad elevada • Portabilidad especialmente elevada si se utiliza la implementación de SQL. • Se proporciona soporte a las rutinas externas de 32 bits y 64 bits en distintos lenguajes de programación	Portabilidad elevada
Interoperatividad	• Los procedimientos pueden llamar a otros procedimientos y contener sentencias de SQL que invoquen a funciones con niveles de acceso a SQL inferiores o iguales al nivel de acceso a SQL del procedimiento.	• Las funciones pueden contener otras sentencias de SQL que invoquen a otras funciones y llamar a procedimientos con niveles de acceso a SQL inferiores o iguales al nivel de acceso a SQL de la función.	• Los métodos pueden invocar funciones con un nivel de acceso a SQL inferior o igual al nivel de acceso a SQL del método. • Los métodos no pueden llamar a procedimientos ni a otros métodos
Restricciones		• Sólo se debe hacer referencia a las funciones de tabla que sólo puedan devolver una única referencia de tabla en la cláusula FROM de una sentencia SELECT. salida.	

En general, las características funcionales y las aplicaciones de las rutinas determinan qué tipo de rutina hay que utilizar. Sin embargo, el rendimiento y las implementaciones de las rutinas soportadas también desempeñan un papel importante al determinar qué tipo de rutina hay que utilizar.

Determinar qué tipo funcional de rutina hay que utilizar

Los procedimientos, funciones y métodos proporcionan un soporte distinto a las características y rutinas funcionales. La determinación del tipo de rutina que hay que utilizar o implementar determinará dónde y cómo se puede hacer referencia e invocar a la funcionalidad de la rutina, influir en las implementaciones de rutina que se pueden utilizar e influir en los tipos de funcionalidad que la rutina puede contener. Si determina qué tipo de rutina es más adecuado para sus necesidades antes de empezar a implementarla, comprobará que ahorra tiempo y posibles frustraciones que pueden surgir más adelante.

Lea acerca de los tipos funcionales de las rutinas para aprender sobre las características de las mismas.

Para determinar si debe utilizarse un procedimiento, una función o un método, realice lo siguiente:

1. Determine la funcionalidad que desea que encapsule la rutina, la interfaz desde la que desea invocar la rutina y la implementación de rutina que desea utilizar.
   - Vea el tema:
     – "Comparación de los tipos funcionales de rutinas" en la página 16

     para determinar qué tipos de rutinas funcionales proporcionan soporte a estos requisitos.
2. Determine las sentencias de SQL que desea incluir en la rutina.
   - Vea el tema:
     – "Sentencias de SQL que pueden ejecutarse en rutinas" en la página 38
   - Determine las rutinas funcionales que proporcionan soporte a la ejecución de las sentencias de SQL necesarias.
3. Si la rutina solo va a incluir una o varias consultas, plantéese la posibilidad de usar funciones de SQL. El rendimiento de las funciones de SQL es muy bueno en esta situación, porque se compilan de manera incorporada con las sentencias de SQL que les hacen referencia, a diferencia de los procedimientos, que se compilan e invocan por separado.
4. Determine si en el futuro es posible que necesite ampliar la funcionalidad de la rutina para incluir la funcionalidad de otro tipo de rutina (por ejemplo, los procedimientos soportan más sentencias de SQL y, en general, más características SQL, que las funciones). Para evitar tener que volver a escribir una función como un procedimiento más adelante, plantéese implementar un procedimiento ahora.

En general, los requisitos funcionales y de SQL motivan la selección del tipo funcional de rutina que hay que implementar. Sin embargo, existen casos en los que es posible crear rutinas equivalentes desde el punto de vista lógico con tipos funcionales distintos. Por ejemplo, es posible volver a escribir la mayoría de procedimientos básicos que devuelven un solo conjunto de resultados como una función de tabla. También es posible volver a escribir los procedimientos básicos con un solo parámetro de salida como funciones escalares.

Una vez haya determinado el tipo funcional de rutina que debe utilizarse, es posible que le interese aprender más acerca de las implementaciones de las rutinas o determinar la implementación de rutina que debe utilizarse.

Implementaciones de las rutinas

Las rutinas puede implementarse de distintas formas. La implementación de una rutina es básicamente la forma subyacente de la rutina que contiene la lógica que se ejecuta cuando se invoca la rutina. Comprender las distintas implementaciones de rutinas soportadas puede ayudarlo a comprender cómo funcionan las rutinas y a determinar qué implementación de rutina elegir al implementar rutinas definidas por el usuario.

Entre las implementaciones de rutinas disponibles se incluyen:

- Rutinas incorporadas
- Rutinas con fuente
- Rutinas SQL
- Rutinas externas

Las rutinas definidas por el sistema pueden implementarse como rutinas incorporadas, rutinas SQL o rutinas externas. Sin embargo, su implementación es esencialmente invisible para el usuario y, en general, debe preocupar poco al usuario.

Las rutinas definidas por el usuario pueden implementarse como rutinas con fuente, rutinas SQL o rutinas externas.

Las características de cada una de estas implementaciones son distintas y puedes dar lugar a un soporte funcional mayor o menor. Antes de decidirse por una implementación en concreto, es buena idea revisar la funcionalidad soportada y las restricciones asociadas con cada implementación leyendo acerca de cada una de las implementaciones y después leyendo sobre el tema:

- "Comparación de implementaciones de rutinas" en la página 29

Una buena comprensión de las implementaciones de las rutinas puede ayudarlo a tomar las decisiones de implementación acertadas así como a depurar y solucionar los problemas de las rutinas existentes.

Implementación de rutinas incorporadas

Las rutinas incorporadas están incorporadas en el código del gestor de bases de datos de DB2. Estas rutinas tienen una tipificación firme y funcionan bien porque su lógica es nativa para el código de base de datos. Estas rutinas se encuentran en el esquema SYSIBM. Algunos ejemplos de funciones escalares y de agregación incorporadas incluyen:

- Funciones escalares incorporadas: +, -, *, /, substr, concat, length, char, decimal, days
- Funciones de agregación incorporadas: avg, count, min, max, stdev, sum, variance

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

Implementación de una rutina con fuente

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

Implementación de rutinas de SQL

Una implementación de rutina de SQL se compone enteramente de sentencias de SQL. Las implementaciones de rutinas de SQL se caracterizan por el hecho de que las sentencias de SQL que definen la lógica de las rutinas están incluidas en la sentencia CREATE utilizada para crear la rutina en la base de datos. Las rutinas de SQL son rápidas y fáciles de implementar a causa de su sintaxis sencilla y su rendimiento es bueno gracias a su estrecha relación con DB2.

El Lenguaje de procedimientos de SQL (SQL PL) es una extensión de lenguaje del SQL básico formado por sentencias y elementos de lenguaje que se pueden utilizar para implementar la lógica de programación en SQL. El SQL PL incluye un conjunto de sentencias para declarar variables y manejadores de condiciones (sentencia DECLARE), asignar valores a las variables (sentencia de asignación) e implementar la lógica procedimental (sentencias de control) tales como IF, WHILE, FOR, GOTO, LOOP, SIGNAL y otras. SQL y SQL PL, o si está limitado, un subconjunto del SQL PL, se puede utilizar para crear procedimientos, funciones y métodos de SQL. Las implementaciones de rutinas de SQL pueden contener sentencias de SQL tradicional así como sentencias de SQL PL, aunque dependiendo del tipo funcional de la rutina, es posible que la ejecución de algunas sentencias de SQL esté restringida.

En general, los procedimientos de SQL proporcionan soporte a más características que las funciones de SQL, incluido un potente mecanismo de gestión de errores que resulta sencillo de implementar. Las rutinas de SQL no pueden realizar llamadas al sistema directamente ni pueden realizar operaciones sobre entidades que residan fuera de la base de datos directamente.

Implementación de rutinas externas

Una implementación de una rutina externa es aquella en la que la lógica de la rutina viene definida por el código del lenguaje de programación que reside fuera de la base de datos. Tal como ocurre con las otras implementaciones de rutinas, las rutinas con implementaciones externas se crean en la base de datos ejecutando una sentencia CREATE. La lógica de la rutina almacenada en una biblioteca compilada reside en el servidor de bases de datos en una vía de acceso de directorio especial. La asociación del nombre de la rutina con la aplicación de código externo está especificada mediante la especificación de la cláusula EXTERNAL en la sentencia CREATE.

Las rutinas externas pueden escribirse en cualquiera de los lenguajes de programación de rutinas externas soportados.

La implementación de rutinas externas puede ser más compleja que la implementación de rutinas SQL. No obstante, resultan muy potentes ya que permiten aprovechar al máximo la funcionalidad y el rendimiento del lenguaje de programación de implementación que se haya elegido. Las funciones externas

también tienen la ventaja de permitir el acceso y la manipulación de entidades que residen fuera de la base de datos, tales como la red o el sistema de archivos. Para las rutinas que requieran un grado menor de interacción con la base de datos DB2, pero que deban contener mucha lógica o lógica muy compleja, la implementación de una rutina externa constituye una buena opción.

Por ejemplo, el uso de las rutinas externas es ideal para implementar nuevas funciones que realicen operaciones con tipos de datos incorporados o amplíen la utilidad de los mismos como, por ejemplo, una nueva función de serie que realice operaciones con un tipo de datos VARCHAR o una función matemática complicada que realice operaciones con un tipo de datos DOUBLE. Las implementaciones de rutinas externas también son ideales para la lógica que puede implicar una acción externa, como enviar un mensaje de correo electrónico.

Si ya se siente cómodo programando en uno de los lenguajes de programación de rutinas externas soportados y necesita encapsular lógica que enfatice más la lógica de programación que el acceso a los datos, una vez aprenda los pasos implicados en la creación de rutinas con implementación externa, no tardará en descubrir la potencia que puede conseguir con las mismas.

Interfaces API y lenguajes de programación soportados para el desarrollo de rutinas externas

Puede desarrollar rutinas externas de DB2 (procedimientos y funciones) utilizando las siguientes API y los lenguajes de programación asociados:

- ADO.NET
  - Lenguajes de programación .NET Common Language Runtime
- CLI
- SQL incorporado
  - C
  - C++
  - COBOL (solo para procedimientos)
- JDBC
  - Java
- OLE
  - Visual Basic
  - Visual C++
  - Cualquier otro lenguaje de programación que soporte esta API.
- OLE DB (solo para funciones de tabla)
  - Cualquier lenguaje de programación que soporte esta API.
- SQLJ
  - Java

Comparación de las interfaces API y los lenguajes de programación soportados para el desarrollo de rutinas externas

Antes de empezar a implementar rutinas externas, es importante tener en cuenta las características y las limitaciones de las distintas interfaces de programación de aplicaciones (API) y lenguajes de programación soportados para las rutinas externas. Con ello se asegurará de que elige la implementación adecuada desde el principio y de que están disponibles las características de rutinas que necesite.

Tabla 3. Comparación de las API y lenguajes de programación de rutinas externas

API y lenguaje de programación	Soporte de características	Rendimiento	Seguridad	Escalabilidad	Limitaciones
SQL (incluye SQL PL)	• SQL es un lenguaje de alto nivel fácil de aprender y de utilizar y que agiliza la implementación. • Los elementos del Lenguaje de procedimientos de SQL (SQL PL) permiten utilizar la lógica de flujo de control en operaciones y consultas SQL. • Soporte a tipos de datos firmes.	• Muy bueno. • El rendimiento de las rutinas SQL es mejor que el de las rutinas Java. • El rendimiento de las rutinas SQL es tan bueno como el de las rutinas externas de C y C++ creadas con la cláusula NOT FENCED.	• Muy seguro. • Los procedimientos de SQL siempre se ejecutan en la misma memoria que el gestor de bases de datos. Esto corresponde a la rutina que se está creando de forma predeterminada con las palabras clave NOT FENCED.	• Gran escalabilidad.	• No pueden acceder al sistema de archivos del servidor de bases de datos. • No pueden invocar aplicaciones que residan fuera de la base de datos.

Tabla 3. Comparación de las API y lenguajes de programación de rutinas externas (continuación)

API y lenguaje de programación	Soporte de características	Rendimiento	Seguridad	Escalabilidad	Limitaciones
SQL incorporado (incluye C y C++)	• Lenguaje de programación de bajo nivel, pero potente.	• Muy bueno. • El rendimiento de las rutinas de C y C++ es mejor que el de las rutinas Java. • El rendimiento de las rutinas de C y C++ creadas con la cláusula NOT FENCED es tan bueno como el de las rutinas SQL.	• Las rutinas de C y C++ son susceptibles a los errores de programación. • Los programadores deben tener un buen dominio del lenguaje C para evitar cometer errores de memoria y de manipulación de punteros muy comunes que hacen que la implementación de las rutinas sea más pesada y requiera más tiempo. • Las rutinas de C y C++ deben crearse con la cláusula FENCED y la cláusula NOT THREADSAFE para evitar la interrupción del gestor de bases de datos en caso de que se produzca una excepción en la rutina en tiempo de ejecución. Éstas son las cláusulas por omisión. La utilización de estas cláusulas puede tener un cierto efecto negativo sobre el rendimiento pero garantiza una ejecución segura. Consulte: Seguridad de las rutinas.	• La escalabilidad es limitada cuando se crean rutinas de C y C++ con las cláusulas FENCED y NOT THREADSAFE. Estas rutinas se ejecutan en un proceso *db2fmp* independiente del proceso del gestor de bases de datos. Se necesita un proceso *db2fmp* para cada rutina que se ejecute de forma simultánea.	• Existen múltiples estilos para pasar los parámetros soportados y esto puede resultar confuso. Los usuarios deben utilizar siempre que sea posible el estilo de los parámetros SQL.

Tabla 3. Comparación de las API y lenguajes de programación de rutinas externas (continuación)

API y lenguaje de programación	Soporte de características	Rendimiento	Seguridad	Escalabilidad	Limitaciones
SQL incorporado (COBOL)	• Lenguaje de programación de alto nivel adecuado para el desarrollo de aplicaciones empresariales que suelen ir orientadas a archivos. • Utilizado de forma generalizada en el pasado para las aplicaciones empresariales de producción, aunque su popularidad está decayendo. • COBOL no contiene soporte a los punteros y es un lenguaje de programación iterativo y lineal.	• El rendimiento de las rutinas de COBOL no es tan bueno como el de las rutinas creadas con el resto de opciones de implementación de rutinas externas.	• No hay información disponible en estos momentos.	• No hay información disponible en estos momentos.	• Es posible crear e invocar procedimientos de COBOL de 32 bits en instancias de DB2 de 64 bits; sin embargo, el rendimiento de estas rutinas no será tan bueno como el de los procedimientos de COBOL de 64 bits de una instancia de DB2 de 64 bits.
JDBC (Java) y SQLJ (Java)	• Lenguaje de programación de alto nivel orientado a objetos adecuado para el desarrollo de aplicaciones autónomas, applets y servlets. • Los objetos y los tipos de datos de Java facilitan el establecimiento de conexiones con la base de datos, la ejecución de las sentencias de SQL y la manipulación de los datos.	• El rendimiento de las rutinas de Java es tan bueno como el de las rutinas de C y C++ o el de las rutinas SQL.	• Las rutinas de Java son más seguras que las rutinas de C y C++ porque la Máquina virtual Java (JVM) es quien gestiona el control de las operaciones peligrosas. Esto aumenta la fiabilidad y hace que resulte difícil que el código de una rutina de Java perjudique a otra rutina que se ejecute en el mismo proceso.	• Buena escalabilidad • Las rutinas Java creadas con la cláusula FENCED THREADSAFE (el valor por omisión) tienen una buena escalabilidad. Todas las rutinas Java protegidas comparten algunas JVM. Es posible que se utilice más de una JVM en el sistema si la pila de Java de un proceso db2fmp concreto está próxima a su agotamiento.	• Para evitar operaciones potencialmente peligrosas, no se permiten llamadas JNI (Java Native Interface) desde las rutinas Java.

Tabla 3. Comparación de las API y lenguajes de programación de rutinas externas (continuación)

API y lenguaje de programación	Soporte de características	Rendimiento	Seguridad	Escalabilidad	Limitaciones
Lenguajes soportados Common Language Runtime (CLR) .NET (incluidos C#, Visual Basic y otros)	• Parte del modelo de código gestionado .NET de Microsoft®. • El código fuente se compila en el bytecode de lenguaje intermedio (IL) que se puede interpretar mediante la unidad ejecutable de lenguaje común (CLR) Microsoft .NET Framework. • Los ensamblajes CLR se pueden crear a partir de subensamblajes que se hayan compilado con diferente código fuente de lenguaje de programación .NET, lo que permite a los usuarios reutilizar e integrar módulos de código escritos en varios lenguajes.	Cuando se crean rutinas que especifican la cláusula LANGUAGE CLR, se debe especificar también la cláusula FENCED. La cláusula FENCED se utiliza por omisión. La cláusula THREADSAFE también se utiliza por omisión, pero se puede especificar la cláusula NOT THREADSAFE. La utilización de los valores de cláusula por omisión minimiza la interrupción del gestor de bases de datos durante la ejecución, pero debido a que las rutinas CLR se deben ejecutar como FENCED, su rendimiento puede ser ligeramente menor que el de rutinas externas con otras implemen-taciones que se pueden especificar como NOT FENCED.	• Cuando se crean rutinas que especifican la cláusula LANGUAGE CLR, se debe especificar también la cláusula FENCED. La cláusula FENCED se utiliza por omisión. La cláusula THREADSAFE también se utiliza por omisión, pero se puede especificar la cláusula NOT THREADSAFE. Estas rutinas son seguras porque se ejecutan fuera del gestor de bases de datos en un proceso db2fmp separado.	• No hay información disponible.	Consulte el tema "Restricciones de las rutinas CLR de .NET".

Tabla 3. Comparación de las API y lenguajes de programación de rutinas externas (continuación)

API y lenguaje de programación	Soporte de características	Rendimiento	Seguridad	Escalabilidad	Limitaciones
• OLE	• Las rutinas de OLE se pueden implantar en Visual C++, en Visual Basic y en otros lenguajes soportados por OLE.	• La velocidad de las rutinas automatizadas OLE dependerá del lenguaje utilizado para implementarlas. En general, son más lentas que las rutinas C/C++ que no son OLE. • Las rutinas OLE solo se pueden ejecutar en modalidad FENCED NOT THREADSAFE y, por lo tanto, las rutinas automatizadas OLE tienen una buena escalabilidad.	• No hay información disponible.	• No hay información disponible.	• No hay información disponible.

Tabla 3. Comparación de las API y lenguajes de programación de rutinas externas (continuación)

API y lenguaje de programación	Soporte de características	Rendimiento	Seguridad	Escalabilidad	Limitaciones
• OLE DB	• OLE DB se puede utilizar para crear funciones de tabla definidas por el usuario. • Las funciones OLE DB establecen conexión con fuentes de datos externas OLE DB.	• El rendimiento de las funciones OLE DB depende del proveedor de OLE DB; sin embargo, en general las funciones OLE DB ofrecen un mejor rendimiento que las funciones Java equivalente en términos lógicos pero menor que las funciones C, C++ o SQL equivalentes en términos lógicos. No obstante, algunos predicados de la consulta en que se invoca la función se pueden evaluar en el proveedor de OLE DB, por lo que quedará reducido el número de filas que DB2 tenga que procesar, lo que suele dar lugar a un mejor rendimiento.	• No hay información disponible.	• No hay información disponible.	• OLE DB sólo se puede utilizar para crear funciones de tabla definidas por el usuario.

Comparación de implementaciones de rutinas

Comprender las diferencias entre las implementaciones de rutinas soportadas puede ayudarlo a determinar qué implementación de rutina utilizar al crear sus propias rutinas. Esto puede ahorrarle tiempo y esfuerzo así como garantizarle la maximización de la funcionalidad y el rendimiento de las rutinas.

Las implementaciones de rutinas incorporadas, con fuente, SQL y externas difieren en una serie de aspectos. Estas diferencias se resaltan en la tabla siguiente:

Tabla 4. Comparación de implementaciones de rutinas

Características	Incorporadas	Con fuente	SQL	Externas
Características y usos	• Soporte a tipos de datos firmes. • Su rendimiento es muy bueno porque su lógica es nativa para el código de base de datos. • Muchas funciones incorporadas habituales de conversión, manipulación de series y aritmética se encuentra en el esquema SYSIBM.	• Soporte a tipos de datos firmes. • Utilizadas para proporciona ampliaciones básicas de la funcionalidad de las funciones incorporadas. • SQL y SQL PL proporcionan soporte al lenguaje de programación de alto nivel, lo que agiliza y simplifica la implementación de la lógica de la rutina.	• Soporte a tipos de datos firmes. • Se utilizan para ampliar el conjunto de funciones incorporadas con funciones más complejas que puedan ejecutar sentencias de SQL.	• Los desarrolladores pueden programar la lógica en el lenguaje de programación soportado que elijan. • Es posible implementar lógica compleja. • Soporte directo a acciones externas, a acciones con un efecto fuera de la base de datos. Esto puede incluir operaciones como leer o escribir en el sistema de archivos del servidor, invocar una aplicación o script del servidor y emitir sentencias de SQL que no estén soportadas en las implementaciones de SQL, con fuente o incorporadas.
¿Está la implementación incorporada en el código del gestor de bases de datos?	• Sí	• No	• No	• No
Tipos de rutinas funcionales soportados que pueden tener esta implementación	• No aplicable	• Funciones – Funciones escalares – Funciones de agregación	• Procedimientos • Funciones • Métodos	• Procedimientos • Funciones • Métodos
Sentencias de SQL soportadas	• No aplicable	• No aplicable	• La mayoría de sentencias de SQL, incluidas las sentencias de SQL PL, pueden ejecutarse en las rutinas. • Consulte el tema "Sentencias de SQL que pueden ejecutarse en rutinas".	• Muchas de las sentencias de SQL, incluido un subconjunto de las sentencias de SQL PL, pueden ejecutarse en las rutinas. • Consulte el tema "Sentencias de SQL que pueden ejecutarse en rutinas".

Tabla 4. Comparación de implementaciones de rutinas (continuación)

Características	Incorporadas	Con fuente	SQL	Externas
Rendimiento	• Muy rápido	• En general, igual de rápidas que las funciones incorporadas.	• Rendimiento muy bueno si el SQL está escrito de forma eficaz, las operaciones de la base de datos se enfatizan más que la lógica de programación y se adoptan las mejores prácticas de las rutinas SQL. Consulte el tema "Mejores prácticas de las rutinas SQL".	• Rendimiento muy bueno si la lógica de programación está escrita de forma eficaz y se adoptan las mejores prácticas de las rutinas SQL. Consulte el tema "Mejores prácticas de las rutinas externas".
Portabilidad	• No aplicable	• Resulta sencillo descartar las funciones con fuente y volver a crearlas en otras bases de datos de DB2.	• Las funciones de SQL pueden descartarse con facilidad y volver a crearse en otras bases de datos.	• Las funciones externas puede descartarse y volver a crearse en otras bases de datos. Sin embargo, debe garantizarse que el entorno sea compatible y que esté disponible el software soportado que sea necesario. Consulte el tema "Despliegue de las rutinas externas".
Interoperatividad	• No aplicable	• Puede hacerse referencia a las mismas en todos aquellos lugares en los que pueda hacerse referencia a las funciones incorporadas. Las funciones con fuente no pueden invocar otras funciones.	• Es posible hacer referencia a las rutinas SQL en muchos lugares de las sentencias de SQL. Una rutina SQL puede invocar otras rutinas SQL y externas con niveles de acceso a SQL que sean iguales o inferiores al nivel de acceso a SQL de la rutina SQL.	• Las rutinas externas pueden invocar rutinas externas y otras rutinas SQL con niveles de acceso a SQL que sean iguales o inferiores al nivel de acceso a SQL de la rutina externa.

En general, las características funcionales y las aplicaciones de las rutinas determinan qué tipo de rutina hay que utilizar. Sin embargo, el rendimiento y las implementaciones de las rutinas soportadas también desempeñan un papel importante al determinar qué tipo de rutina hay que utilizar.

Determinar qué implementación de rutina hay que utilizar

La elección de utilizar o crear una rutina con una implementación incorporada, con origen, SQL o externa puede influir en qué función proporcionará la rutina, en el rendimiento de la rutina y en la probabilidad de que, en el momento de la ejecución, surjan problemas que exijan depuración.

Siempre que sea posible, si hay una rutina definida por el sistema que proporcione el soporte que se necesita, utilícela. Utilice las rutinas incorporadas existentes siempre que sea posible. Si la función que necesita es muy parecida a la de una función incorporada existente, plantéese la posibilidad de crear una función con origen que la amplíe.

Si debe crear una rutina, siga el procedimiento que figura más abajo. Es importante determinar qué implementación de rutina hay que usar antes de avanzar demasiado en el diseño de la rutina.

Para determinar si hay que utilizar una implementación de rutina con origen, SQL o externa al crear una rutina, haga lo siguiente:

1. Determine si desea crear un procedimiento, una función o un método. Este debe ser siempre el primer paso al desarrollar una rutina. Determine también cuáles son las implementaciones de soporte para ese tipo de rutina. Consulte:
   - "Comparación de los tipos funcionales de rutinas" en la página 16
2. Determine las sentencias de SQL que desea incluir en la rutina. El conjunto de sentencias de SQL que desea ejecutar en una rutina puede limitar la elección de implementación de la rutina. Consulte:
   - "Determinar qué sentencias de SQL se pueden ejecutar en la rutina" en la página 44
3. Determine si, ahora o en el futuro, la lógica de la rutina debe acceder a datos, archivos o aplicaciones que residan en un lugar externo a la base de datos. Los datos, archivos o aplicaciones podrían residir en el sistema de archivos del servidor de bases de datos o en la red disponible.
   - Si la lógica de la rutina debe acceder a entidades situadas fuera de la base de datos, hay que usar una implementación de rutina externa.
4. Determine el número de consultas que hay que incluir en la rutina en relación con la cantidad de lógica de flujo de procedimientos.
   - Si la lógica de la rutina contiene principalmente lógica de flujo de procedimientos y muy pocas consultas, cree una rutina externa.
   - Si la lógica de la rutina contiene muchas consultas y una cantidad mínima de lógica de flujo de procedimientos, cree una rutina SQL.

Uso de las rutinas

Las rutinas pueden utilizarse para solucionar muchos de los problemas habituales que encuentra tanto los diseñadores de bases de datos, como los administradores de bases de datos y los desarrolladores de aplicaciones. Pueden ayudar a mejorar la estructura, mantenimiento y rendimiento de las aplicaciones. Estos son algunos ejemplos de escenarios en los que podría utilizar rutinas:

- Administración de bases de datos con rutinas
- Ampliación del soporte a las funciones de SQL con las funciones definidas por el usuario
- Auditoría de las modificaciones de los datos utilizando rutinas y otras características de SQL

Administración de bases de datos con rutinas definidas por el sistema

Resulta posible administrar bases de datos mediante aplicaciones y con la introducción de las rutinas definidas por el sistema que tienen como finalidad explícita la realización de funciones administrativas se ha facilitado esta tarea. A partir de la versión 8.1, DB2 proporciona un conjunto de procedimiento y de funciones definidos por el sistema en los esquemas SYSPROC, SYSFUN y SYSTOOLS que están listos para su utilización para realizar tareas administrativas incluida la ejecución de mandatos de DB2 mediante una interfaz de SQL, la modificación de los parámetros de configuración, la gestión de paquetes, las tareas relacionadas con instantáneas y más. Puede optar por utilizar las rutinas administrativas definidas por el sistema si necesita una aplicación para realizar tareas administrativas o si desea acceder a los resultados de las tareas administrativas mediante una interfaz de SQL para poder filtrar, clasificar, modificar o reutilizar los resultados en otra consulta y si no desea crear sus propias rutinas para ello.

A partir de DB2 Universal Database versión 9.1 se incluye una nueva rutina administrativa definida por el sistema denominada ADMIN_CMD. Ésta, junto con las muchas otras rutinas definidas por el sistema, proporcionan amplio soporte administrativo.

ADMIN_CMD para invocar los mandatos de DB2 mediante una interfaz de SQL

A partir de la versión 9.1, se incluye una nueva rutina administrativa definida por el sistema denominada ADMIN_CMD que permite ejecutar mandatos de DB2 mediante una interfaz de SQL. Básicamente, esta rutina permite pasar como un argumento un mandato de DB2 con los distintivos y valores adecuados como un parámetro de serie. La rutina ejecuta la serie que contiene el mandato de DB2 y devuelve los resultados en formato tabular o escalar, que puede utilizarse como parte de una consulta u operación mayor. Esta funcionalidad hace que sea más fácil que nunca escribir aplicaciones de bases de datos administrativas.

Rutinas administrativas definidas por el sistema

Entre los ejemplos de otras rutinas definidas por el sistema se incluyen: SNAPSHOT_TABLE, HEALTH_DB_HI, SNAPSHOT_FILEW, REBIND_ROUTINE_PACKAGE. Éstas y muchas otras rutinas definidas por el sistema pueden utilizarse desde el CLP o en aplicaciones de bases de datos siempre que se proporcione soporte a la invocación de la rutina especificada.

La rutina ADMIN_CMD y otras rutinas definidas por el sistema están disponibles para su utilización siempre que el nombre de los esquemas SYSPROC y SYSFUN esté incluido en el valor CURRENT PATH, como lo está por omisión.

Si desea ver ejemplos de la utilización de rutinas definidas por el usuario, consulte la documentación de consulta específica sobre las rutinas definidas por el usuario.

Ampliación del soporte a las funciones de SQL con las funciones definidas por el usuario

Si ninguna de las funciones definidas por el sistema encapsula la lógica que necesita, puede crear sus propias funciones definidas por el usuario. Las funciones

definidas por el usuario constituyen una forma buena para ampliar el conjunto básico de funciones de SQL. Si usted o un grupo de usuarios necesitan una función para implementar una fórmula matemática compleja, una manipulación determinada de una serie o para realizar transformaciones semánticas de valores, es posible crear para ello, con facilidad, una función de SQL de alto rendimiento, a la que puede hacerse referencia como cualquier función de SQL incorporado existente.

Por ejemplo, supongamos que un usuario necesita una función que convierta un valor de una divisa en otra divisa. Esta función no está disponible en el conjunto de rutinas definidas por el sistema. Sin embargo, sí que puede crearse como una función escalar de SQL definida por el usuario. Una vez creada, podrá hacerse referencia a esta función siempre que se proporcione soporte a las funciones escalares en una sentencia de SQL.

Otro usuario podría necesitar una función más compleja que enviara una correo electrónico siempre que se realice una modificación en una columna concreta de una tabla. Esta función no está disponible en el conjunto de rutinas definidas por el sistema. Sin embargo, sí que puede crearse como un procedimiento externo definido por el usuario con una implementación del lenguaje de programación C. Una vez creado, podrá hacerse referencia a este procedimiento siempre que se proporcione soporte a los procedimientos, también desde activadores.

Estos ejemplos demuestran la facilidad con la que es posible ampliar el lenguaje de SQL creando rutinas definidas por el usuario.

Auditoría mediante funciones de tabla de SQL

Los administradores de bases de datos interesados en supervisar accesos y modificaciones de datos de tabla efectuados por los usuarios de las bases de datos, pueden comprobar las transacciones sobre una tabla creando y utilizando funciones de tabla de SQL que modifican datos de SQL.

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

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

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

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

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

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

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

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

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

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

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

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

```
        FROM FINAL TABLE(UPDATE employee
                         SET salary = salary + amount
                         WHERE employee.empnum = updEmpNum);
  END
```

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

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

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

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

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

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

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

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

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

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

Herramientas para desarrollar rutinas

Hay disponibles varios entornos y herramientas de desarrollo que permiten desarrollar procedimientos y funciones. Algunas de estas herramientas se proporcionan con DB2 Database para Linux®, UNIX® y Windows® mientras que otras son componentes integrados en entornos de desarrollo integrados habituales. Hay interfaces gráficas y no gráficas y herramientas que pueden utilizarse para desarrollar procedimientos y funciones.

La siguiente herramienta de interfaz gráfica de usuario (GUI), proporcionada con DB2, está disponible para el desarrollo de rutinas en servidores de bases de datos de DB2:

- IBM® Data Studio

La siguiente interfaz de línea de mandatos, proporcionada con DB2, está disponible para el desarrollo de rutinas en servidores de bases de datos de DB2:

- Procesador de línea de mandatos de DB2 (DB2 CLP)

Varios de los productos de software de IBM proporcionan herramientas gráficas para el desarrollo de rutinas en los servidores de bases de datos de DB2 incluidos, entre otros:

- IBM Distributed Unified Debugger
- IBM Rational Application Developer
- IBM Rational Web Developer
- IBM WebSphere Studio

Varios productos de software de fuentes abiertas proporcionan herramientas gráficas para el desarrollo de rutinas en los servidores de bases de datos de DB2 incluidos, entre otros:

- Herramientas Web de DB2 para Eclipse Framework

Algunas características de DB2 pueden utilizarse para añadir soporte a las herramientas gráficas para el desarrollo de rutinas en software proporcionado por otros proveedores, entre el que se incluye:

- IBM Database Add-Ins para Microsoft Visual Studio

No hay herramientas disponibles para desarrollar métodos.

Soporte para el desarrollo de rutinas de IBM Data Studio

La IBM Data Studio proporciona un entorno de desarrollo de empleo sencillo para crear, construir, depurar, probar y desplegar procedimientos almacenados. Data Studio ofrece herramientas gráficas que simplifican el proceso de creación de rutinas permitiéndole concentrarse en la lógica del procedimiento almacenado en vez de hacerlo en los detalles de la generación de la sentencia CREATE básica, la creación e instalación de procedimientos almacenados en un servidor DB2. Además, con la herramienta del desarrollador, puede desarrollar procedimientos almacenados en un sistema operativo y crearlos en otros sistemas operativos de servidor.

La herramienta del desarrollador es una aplicación gráfica que da soporte a un desarrollo rápido. Con la herramienta del desarrollador podrá realizar las siguientes tareas:

- Crear rutinas nuevas.

- Crear rutinas en servidores DB2 locales y remotos.
- Modificar y volver a crear rutinas existentes.
- Probar y depurar la ejecución de rutinas instaladas.

Data Studio se instala desde un DVD que está incluido en DB2 Database para Linux, UNIX y Windows. Después de instalar Data Studio, podrá iniciarlo desde las ubicaciones siguientes:

- En Windows: pulse **Inicio** > **Todos los programas** > **IBM Software Development Platform** > **IBM Data Studio** > **IBM Data Studio**
- En Linux: pulse **Programming** > **IBM Data Studio** o **Development** > **IBM Data Studio** según su versión de Linux.

También se puede iniciar Data Studio desde el Centro de control para DB2 para OS/390. Data Studio puede iniciarse como un proceso independiente desde el menú Herramientas del Centro de control, desde la barra de herramientas o desde la carpeta Procedimientos almacenados. Además, desde la ventana Proyecto de Data Studio se puede exportar uno o más procedimientos almacenados de SQL creados para un servidor DB2 para OS/390 a un archivo específico que se puede ejecutar en el procesador de línea de mandatos (CLP).

Data Studio permite gestionar su trabajo en proyectos. Cada proyecto de Data Studio guarda sus conexiones con bases de datos específicas, como por ejemplo, servidores DB2 para OS/390. Además, puede crear filtros para visualizar subconjuntos de los procedimientos almacenados en cada una de las bases de datos. Cuando abra un proyecto nuevo o existente de Data Studio, puede filtrar los procedimientos almacenados de forma que se visualicen en base a su nombre, esquema, lenguaje o ID de colección (sólo para OS/390).

Sentencias de SQL que pueden ejecutarse en rutinas

La ejecución satisfactoria de sentencias de SQL en rutinas está sujeta a restricciones y tiene como condición que se cumplan ciertos prerrequisitos; no obstante, es posible ejecutar muchas sentencias de SQL en rutinas.

Si una sentencia invoca una rutina, el nivel de acceso a datos de SQL efectivo para la sentencia será el mayor de:

- El nivel de acceso a datos de SQL de la sentencia de la tabla siguiente.
- El nivel de acceso a datos de SQL de la rutina especificada cuando se creó la rutina.

Por ejemplo, una sentencia CALL tiene un nivel de acceso a datos de SQL de CONTAINS SQL. Sin embargo, si se invoca un procedimiento almacenado definido como READS SQL DATA, el nivel de acceso a datos de SQL efectivo para la sentencia CALL será READS SQL DATA.

La tabla siguiente lista todas las sentencias de SQL soportadas, incluidas sentencias de control de SQL PL e identifica si puede ejecutar cada sentencia de SQL dentro de los diversos tipos de rutinas. Para cada sentencia de SQL listada en la primera columna, en cada una de las columnas subsiguientes que identifican los tipos de rutina e implementaciones soportados, una X indica si la sentencia es ejecutable dentro de la rutina. Una columna aparte identifica el nivel de acceso a SQL mínimo necesario que debe especificarse para que se permita la ejecución de la sentencia. A menos que se indique lo contrario en una nota a pie de página, todas las sentencias de SQL pueden ejecutarse tanto de manera estática como dinámica.

Tabla 5. Sentencias de SQL que pueden ejecutarse en rutinas

Sentencia de SQL	Ejecutable en procedimientos de SQL	Ejecutable en funciones de SQL	Ejecutable en procedimientos externos	Ejecutable en funciones externas	Nivel de acceso a datos de SQL mínimo necesario
ALLOCATE CURSOR	X		X	X	MODIFIES SQL DATA
ALTER {BUFFERPOOL, DATABASE PARTITION GROUP, FUNCTION, METHOD, NICKNAME, PROCEDURE, SEQUENCE, SERVER, TABLE, TABLESPACE, TYPE, USER MAPPING, VIEW}			X	X	MODIFIES SQL DATA
ASSOCIATE LOCATORS	X				
AUDIT					MODIFIES SQL DATA
BEGIN DECLARE SECTION			X	X	NO SQL(1)
CALL	X	X	X	X	READS SQL DATA
CASE	X				CONTAINS SQL
CLOSE	X		X	X	READS SQL DATA
COMMENT ON	X		X	X	MODIFIES SQL DATA
COMMIT	X(4)		X(4)		MODIFIES SQL DATA
SQL compuesto	X	X	X	X	CONTAINS SQL
CONNECT(2)					

Tabla 5. Sentencias de SQL que pueden ejecutarse en rutinas (continuación)

Sentencia de SQL	Ejecutable en procedimientos de SQL	Ejecutable en funciones de SQL	Ejecutable en procedimientos externos	Ejecutable en funciones externas	Nivel de acceso a datos de SQL mínimo necesario
CREATE {ALIAS, BUFFERPOOL, DATABASE PARTITION GROUP, DISTINCT TYPE, EVENT MONITOR, FUNCTION, FUNCTION MAPPING, INDEX, INDEX EXTENSION, METHOD, NICKNAME, PROCEDURE, SCHEMA, SEQUENCE, SERVER, TABLE, TABLESPACE, TRANSFORM, TRIGGER, TYPE, TYPE MAPPING, USER MAPPING, VIEW, WRAPPER }	X (6)		X		MODIFIES SQL DATA
DECLARE CURSOR	X		X		NO SQL(1)
DECLARE GLOBAL TEMPORARY TABLE	X		X	X	MODIFIES SQL DATA
DELETE	X	X	X	X	MODIFIES SQL DATA
DESCRIBE(7)			X	X	READS SQL DATA
DISCONNECT(2)					
DROP	X(6)		X	X	MODIFIES SQL DATA
END DECLARE SECTION			X	X	NO SQL(1)
EXECUTE	X		X	X	CONTAINS SQL(3)
EXECUTE IMMEDIATE	X		x	X	CONTAINS SQL(3)
EXPLAIN	X		X	X	MODIFIES SQL DATA
FETCH	X		X	X	READS SQL DATA
FREE LOCATOR	X	X	X	X	CONTAINS SQL

Tabla 5. Sentencias de SQL que pueden ejecutarse en rutinas (continuación)

Sentencia de SQL	Ejecutable en procedimientos de SQL	Ejecutable en funciones de SQL	Ejecutable en procedimientos externos	Ejecutable en funciones externas	Nivel de acceso a datos de SQL mínimo necesario
FLUSH EVENT MONITOR			X	X	MODIFIES SQL DATA
FLUSH PACKAGE CACHE			X	X	MODIFIES SQL DATA
FOR	X	X			READS SQL DATA
FREE LOCATOR	X		X	X	CONTAINS SQL
GET DIAGNOSTICS	X	X			READS SQL DATA
GOTO	X				CONTAINS SQL
GRANT	X		X	X	MODIFIES SQL DATA
IF	X	X			CONTAINS SQL
INCLUDE			X	X	NO SQL
INSERT	X	X	X	X	MODIFIES SQL DATA
ITERATE	X	X			CONTAINS SQL
LEAVE	X	X			CONTAINS SQL
LOCK TABLE	X		X	X	CONTAINS SQL
LOOP	X				CONTAINS SQL
MERGE	X	X	X	X	MODIFIES SQL DATA
OPEN	X		X	X	READS SQL DATA(5)
PREPARE	X		X	X	CONTAINS SQL
REFRESH TABLE			X	X	MODIFIES SQL DATA
RELEASE(2)					
RELEASE SAVEPOINT	X		X	X	MODIFIES SQL DATA
RENAME TABLE			X	X	MODIFIES SQL DATA
RENAME TABLESPACE			X	X	MODIFIES SQL DATA
REPEAT	X				CONTAINS SQL
RESIGNAL	X				MODIFIES SQL DATA
RETURN	X				CONTAINS SQL
REVOKE			X	X	MODIFIES SQL DATA
ROLLBACK(4)	X		X		

Tabla 5. Sentencias de SQL que pueden ejecutarse en rutinas (continuación)

Sentencia de SQL	Ejecutable en procedimientos de SQL	Ejecutable en funciones de SQL	Ejecutable en procedimientos externos	Ejecutable en funciones externas	Nivel de acceso a datos de SQL mínimo necesario
ROLLBACK TO SAVEPOINT	X		X	X	MODIFIES SQL DATA
SAVEPOINT	X				MODIFIES SQL DATA
Sentencia-select	X		X	X	READS SQL DATA
SELECT INTO			X(8)	X(8)	READS SQL DATA(5)
SET CONNECTION(2)					
SET INTEGRITY			X		MODIFIES SQL DATA
registro especial SET	X	X	X	X	CONTAINS SQL
variable SET	X	X			CONTAINS SQL
SIGNAL	X	X			MODIFIES SQL DATA
TRANSFER OWNERSHIP					MODIFIES SQL DATA
UPDATE	X	X	X		MODIFIES SQL DATA
VALUES INTO	X		X	X	READS SQL DATA
WHENEVER	X		X		NO SQL(1)
WHILE	X	X			

Nota:

1. Aunque la opción NO SQL implica que no puede especificarse ninguna sentencia de SQL, las sentencias no ejecutables no están restringidas.
2. Las sentencias de gestión de conexiones no se permiten en ningún contexto de ejecución de rutinas.
3. Dependerá de la sentencia que esté ejecutándose. La sentencia especificada para la sentencia EXECUTE debe ser una sentencia permitida en el contexto del nivel de acceso de SQL concreto en vigor. Por ejemplo, si el nivel de acceso SQL READS SQL DATA está en vigor, la sentencia no puede ser ni INSERT, UPDATE ni DELETE.
4. La sentencia COMMIT y la sentencia ROLLBACK sin la cláusula TO SAVEPOINT puede utilizarse en un procedimiento almacenado, pero únicamente en el caso de que el procedimiento almacenado se llame directamente desde una aplicación o indirectamente por medio de un procedimiento almacenado anidado desde una aplicación. (Si en la cadena de llamada del procedimiento almacenado hay alguna sentencia compuesta atómica, activador, función o método, no se permitirán las acciones COMMIT o ROLLBACK de una unidad de trabajo).

5. Si el nivel de acceso READS SQL DATA está en vigor, ninguna sentencia de cambio de datos SQL podrá estar incorporada en la sentencia SELECT INTO o en el cursor al que se hace referencia mediante la sentencia OPEN.
6. Los procedimientos de SQL sólo pueden emitir las sentencias CREATE y DROP para índices, tablas y vistas.
7. La sentencia DESCRIBE de SQL tiene una sintaxis diferente que la del mandato DESCRIBE de CLP.
8. Solamente se puede utilizar en rutinas de SQL incorporado.

Tabla 5 en la página 39 indica si una sentencia de SQL (especificada en la primera columna) tiene permitido o no ejecutarse en una rutina con la indicación de acceso a datos de SQL especificada. Si se encuentra una sentencia de SQL ejecutable en una rutina definida con NO SQL, se devuelve SQLSTATE 38001. Para otros contextos de ejecución, las sentencias de SQL que no se soportan en ningún contexto devuelven SQLSTATE 38003. Para otras sentencias de SQL no permitidas en un contexto CONTAINS SQL, se devuelve SQLSTATE 38004. En un contexto READS SQL DATA, se devuelve SQLSTATE 38002. Durante la creación de una rutina SQL, una sentencia que no coincida con la indicación de acceso a datos SQL haría que se devolviera SQLSTATE 42985.

Niveles de acceso de SQL en rutinas

El grado al que las rutinas pueden ejecutar sentencias de SQL se determina mediante el nivel de acceso de SQL de la rutina. El nivel de acceso de SQL correspondiente a una rutina se determina tanto por lo que está permitido para el tipo determinado de rutina como por la limitación especificada explícitamente dentro de la sentencia CREATE que define la rutina.

Los niveles de acceso de SQL son los siguientes:

- NO SQL
- CONTAINS SQL
- READS SQL
- MODIFIES SQL

Esta cláusula de nivel de acceso de SQL se emplea para proporcionar información al gestor de bases de datos sobre la sentencia a fin de que el gestor de bases de datos pueda ejecutar dicha sentencia de forma segura y con el mejor rendimiento posible.

Los niveles de acceso de SQL máximo y por omisión para los distintos tipos de rutinas son los siguientes:

Tabla 6. Niveles de acceso de SQL por omisión y máximo para rutinas

Tipo de rutina	**Nivel de acceso de SQL por omisión**	**Nivel de acceso de SQL máximo permitido**
procedimientos de SQL	MODIFIES SQL DATA	MODIFIES SQL DATA
Funciones de SQL (funciones escalares)	READS SQL DATA	READS SQL DATA
Funciones de SQL (funciones de tabla)	READS SQL DATA	MODIFIES SQL DATA
Procedimientos externos	MODIFIES SQL DATA	MODIFIES SQL DATA
Funciones externas (funciones escalares)	READS SQL DATA[1]	READS SQL DATA

Tabla 6. Niveles de acceso de SQL por omisión y máximo para rutinas (continuación)

Tipo de rutina	Nivel de acceso de SQL por omisión	Nivel de acceso de SQL máximo permitido
Funciones externas (funciones de tabla)	READS SQL DATA[1]	READS SQL DATA

Nota:

1. En los entornos SMP y DPF, debe especificarse NO SQL para las funciones externas.

Se consigue un rendimiento óptimo de las rutinas cuando se especifica la cláusula de acceso de SQL más restrictiva que sea válida en la sentencia CREATE de la rutina.

En la sentencia CREATE correspondiente a una rutina:

- Si se especifica de forma explícita READS SQL DATA, ninguna sentencia de SQL de la rutina podrá modificar datos.
- Si se especifica de forma explícita CONTAINS SQL DATA, ninguna sentencia de SQL de la rutina podrá modificar ni leer datos.
- Si se especifica de forma explícita NO SQL, no deberá haber ninguna sentencia de SQL ejecutable en la rutina.

Determinar qué sentencias de SQL se pueden ejecutar en la rutina

Muchas de las sentencias de SQL pueden ejecutarse en las rutinas, pero no todas. La ejecución de una sentencia de SQL concreta en una rutina depende del tipo de rutina, la implementación de la rutina, el nivel máximo de acceso a SQL especificado para la rutina y los privilegios de quien define e invoca la rutina.

El hecho de determinar qué sentencias de SQL se pueden ejecutar en una rutina antes de su implementación puede garantizar que se ha hecho la elección adecuada del tipo de rutina y de la implementación desde el principio.

Para ejecutar satisfactoriamente una sentencia de SQL en una rutina, debe satisfacerse los prerrequisitos siguientes:

- El nivel de acceso a SQL de la rutina debe permitir la ejecución de la sentencia de SQL en concreto.
  - El nivel de acceso a SQL de una rutina está especificado en la sentencia CREATE de la rutina.
  - No se proporciona soporte a algunos niveles de acceso a SQL para ciertos tipos de rutinas. Consulte las restricciones a continuación.
- Quien define la rutina debe tener los privilegios necesarios para ejecutar la sentencia de SQL.
  - En la publicación de consulta SQL se proporcionan los privilegios necesarios para ejecutar cada una de las sentencias de SQL soportadas.
- Ninguna otra restricción distinta limita la ejecución de la sentencia.
  - En la publicación de consulta SQL hallará una lista de las restricciones específicas para cada sentencia de SQL en concreto.

Las restricciones siguientes limitan el conjunto de sentencias de SQL que pueden ejecutarse en las rutinas. En particular, estas restricciones limitan los niveles de acceso a SQL que pueden especificarse para tipos de rutinas concretos.

- No es posible especificar funciones externas con el nivel de acceso MODIFIES SQL DATA.
- Los procedimientos externos a los que un activador deba llamar no pueden especificarse con un nivel de acceso MODIFIES SQL DATA.

Para determinar qué sentencias de SQL se pueden invocar en una rutina concreta, haga lo siguiente:

1. Determine el nivel de acceso a SQL de la rutina. Si se trata de una rutina existente, examine la sentencia CREATE que se haya utilizado para crear la rutina. Es posible que la cláusula de nivel de acceso a SQL esté definida de forma explícita en la DDL con uno de los valores siguientes: NO SQL, CONTAINS SQL, READS SQL DATA o MODIFIES SQL DATA. Si no se ha definido de forma explícita esta cláusula, se asume el valor por omisión para la rutina.
   - Para los procedimientos de SQL, el valor por omisión es MODIFIES SQL DATA.
   - Para las funciones de SQL el valor por omisión es MODIFIES SQL DATA.
   - Para los procedimientos externos, el valor por omisión es MODIFIES SQL DATA.
   - Para las funciones externas, el valor por omisión es READS SQL DATA.
2. Consulte la tabla que se incluye en el tema "Sentencias SQL que pueden ejecutarse en rutinas". Busque la sentencia de SQL que le interese por nombre.
3. Compruebe si se proporciona soporte a la sentencia de SQL para el tipo de rutina e implementación en concreto.
4. Verifique que el nivel de acceso a SQL necesario para ejecutar la sentencia coincida con el nivel de acceso a SQL de la rutina.
5. Lea con atención las notas de uso o las notas a pie de página para asegurarse de que no existan otras restricciones sobre la ejecución de la sentencia de SQL.

Si se indica que la sentencia de SQL se puede ejecutar en una rutina, que el nivel de acceso a SQL de la rutina satisface los prerrequisitos para ejecutar la sentencia en la rutina, y que todos los otros prerrequisitos se han satisfecho, la sentencia de SQL se debe ejecutar de forma satisfactoria desde la rutina.

Portabilidad de las rutinas

La portabilidad de las rutinas hace referencia a la facilidad con la que puede desplegarse una rutina. La portabilidad consta de factores como la compatibilidad del sistema operativo, la compatibilidad del entorno de ejecución, la compatibilidad del software y la compatibilidad de la interfaz de invocación, así como otros factores para la implementación de rutinas como la compatibilidad del soporte de las sentencias de SQL ejecutadas en una rutina.

La portabilidad de las rutinas es esencial si el entorno en el que se desplegará una rutina no es idéntico al entorno en el que se ha desarrollado la rutina. En general, las rutinas de DB2 presentan una portabilidad elevada entre los sistemas operativos e incluso entre los distintos productos y ediciones de base de datos DB2. Es una buena idea pensar en los posibles problemas de portabilidad antes de empezar a desarrollar las rutinas para minimizar la probabilidad de deber rehacer el trabajo más tarde.

Los temas siguientes incluyen información relacionada con factores que pueden limitar la portabilidad de las rutinas:

- Ediciones de DB2 Database para Linux, UNIX y Windows soportadas.
- Software de compilador y desarrollo soportado
- Sentencias de SQL que pueden ejecutarse en rutinas
- Restricciones de las rutinas
- Despliegue de las rutinas

Interoperabilidad de las rutinas

La interoperatividad de rutinas de distintos tipos y con implementaciones de programación diferentes garantiza que las rutinas puedan ser módulos altamente reutilizables mientras dure un sistema de bases de datos. Puesto que son distintos programadores, con experiencia en la programación de lenguajes de programación diferentes, los que a menudo implementan los módulos de código y como suele ser aconsejable reutilizar el código siempre que sea posible para ahorrar tiempo y costes de desarrollo, la infraestructura de las rutinas de DB2 está diseñada para proporcionar soporte a una elevada interoperatividad de las rutinas.

La interoperatividad de las rutinas se caracteriza por la posibilidad de hacer referencia e invocar rutinas de tipos e implementaciones distintos de otras rutinas de forma sencilla y sin requisitos adicionales. Las rutinas de DB2 son interoperativas de las formas siguientes:

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

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

Rendimiento de las rutinas

El rendimiento de las rutinas se ve afectado por una serie de factores incluidos, entre otros, el tipo y la implementación de la rutina, el número de sentencias de SQL de la rutina, el grado de complejidad de SQL de la rutina, el número de parámetros de la rutina, la eficacia de la lógica en la implementación de la rutina y la gestión de errores de la rutina. Como los usuarios a menudo optan por mejorar el rendimiento de las aplicaciones, es importante sacar el máximo partido del rendimiento de la rutina.

La tabla siguiente resalta algunos de los factores generales que afectan el rendimiento de la rutina y ofrece recomendaciones sobre cómo mejorar el rendimiento de la rutina modificando cada uno de los factores. Para obtener más

detalles sobre los factores de rendimiento que afectan a tipos de rutinas concretos, consulte los temas sobre rendimiento y ajuste para el tipo de rutina concreto.

Tabla 7. Consideraciones sobre el rendimiento y recomendaciones sobre el rendimiento de las rutinas

Consideraciones sobre el rendimiento	**Recomendación para el rendimiento**
Tipo de rutina: procedimiento, función, método	• Los procedimientos, las funciones y los métodos sirven para fines distintos y se hace referencia a los mismos en lugares diferentes. Las diferencias funcionales entre los mismos hacen difícil comparar su rendimiento directamente. • En general, los procedimientos pueden reescribirse en ocasiones como funciones (especialmente si devuelven un valor escalar y sólo consultan los datos) y presentan ligeras mejoras en el rendimiento; sin embargo, estas ventajas suelen ser el resultado de la simplificación del SQL necesario para implementar la lógica de SQL. • Las funciones definidas por el usuario con inicializaciones complejas pueden utilizar áreas reutilizables para almacenar los valores necesarios durante la primera invocación de forma que éstos puedan utilizarse en las invocaciones posteriores.
Implementación de rutinas: definidas por el sistema o definidas por el usuario	• Para una lógica equivalente, las rutinas incorporadas son las que mejor rendimiento presentan, seguidas por las rutinas definidas por el sistema, porque gozan de una relación más cercana con el motor de la base de datos que las rutinas definidas por el usuario. • El rendimiento de las rutinas definidas por el usuario puede ser muy bueno si están bien codificadas y siguen las mejores prácticas.
Implementación de rutinas: SQL o implementación de rutinas externas	• Las rutinas SQL son más eficientes que las rutinas externas porque es el servidor de base de datos de DB2 quien las ejecuta. • Los procedimientos de SQL suelen tener un rendimiento mejor que los procedimientos externos con una lógica equivalente. • Para una lógica sencilla, el rendimiento de una función de SQL es comparable al de una función externa equivalente. • Para lógicas complejas como, por ejemplo, los algoritmos matemáticos y las funciones de manipulación de series, que necesiten escaso SQL es mejor utilizar una rutina externa en un lenguaje de programación de bajo nivel como C, ya que depende en menor grado del soporte de SQL. • Consulte la comparación de las implementaciones de rutinas para ver una comparación de las características, incluido el rendimiento, de las opciones del lenguaje de programación de las rutinas externas soportadas.

Tabla 7. Consideraciones sobre el rendimiento y recomendaciones sobre el rendimiento de las rutinas (continuación)

Consideraciones sobre el rendimiento	Recomendación para el rendimiento
Lenguaje de programación de las implementaciones de rutinas externas	• Consulte: Comparación de API de rutinas externas y lenguajes de programación para ver una comparación de las características de rendimiento que deben tenerse en cuenta al seleccionar la implementación de una rutina externa. • Java (API JDBC y SQLJ) – Las rutinas de Java que requieren mucha memoria se crean mejor si se especifica la cláusula FENCED NOT THREADSAFE. Las rutinas de Java que requieren una memoria media pueden especificarse con la cláusula FENCED THREADSAFE. – Para las invocaciones de rutinas Java protegidas con seguridad de hebras, DB2 intenta elegir un proceso Java de hebras de modalidad protegida con un almacenamiento dinámico de Java que sea lo suficientemente grande como para ejecutar la rutina. Si no es posible identificar grandes consumidores de almacenamiento dinámico en su propio proceso, puede resultar que se produzcan errores de almacenamiento dinámico fuera de Java durante los procesos Java db2fmp de varias hebras. Las rutinas FENCED THREADSAFE, por el contrario, tienen un rendimiento superior ya que pueden compartir un número pequeño de JVM. • C y C++ – En general, las rutinas de C y C++ tienen un rendimiento superior al de otras implementaciones de rutinas externas, así como las rutinas de SQL. – Para conseguir el máximo rendimiento, las rutinas de C y C++ deben compilarse en formato de 32 bits si se desplegarán en una instancia de DB2 de 32 bits y en formato de 64 bits si se desplegarán en una instancia de DB2 de 64 bits. • COBOL – En general, el rendimiento de COBOL es satisfactorio, aunque COBOL no es una de las implementaciones de rutinas que se recomiendan.
Número de sentencias de SQL de la rutina	• Las rutinas deben contener más de una sentencia de SQL; de lo contrario, el coste necesario para invocar la rutina hace que su rendimiento no sea eficaz. • La mejor lógica para su encapsulación en la rutina es aquella que debe realizar varias consultas de base de datos y procesar los resultados intermedios y devolver al final un subconjunto de los datos que se han utilizado. La minería de datos compleja y las actualizaciones importantes que requieren la consulta de datos relacionados son ejemplos de este tipo de lógica. En el servidor de bases de datos se realiza un proceso de SQL considerable y sólo se pasa a quien llama el conjunto de resultados de datos más pequeño.

Tabla 7. Consideraciones sobre el rendimiento y recomendaciones sobre el rendimiento de las rutinas (continuación)

Consideraciones sobre el rendimiento	Recomendación para el rendimiento
Complejidad de las sentencias de SQL de la rutina	• Resulta sensato incluir en las rutinas las consultas que son muy complejas para poder beneficiarse de la mayor capacidad de memoria y de rendimiento del servidor de bases de datos. • No se preocupe de si las sentencias de SQL son excesivamente complejas.
Ejecución de SQL estático y dinámico con rutinas	• En general, el rendimiento del SQL estático es mejor que el del SQL dinámico. En las rutinas, no hay diferencias adicionales al utilizar SQL estático o dinámico.
Número de parámetros de las rutinas	• Al minimizar el número de parámetros de las rutinas puede mejorarse el rendimiento de las rutinas, ya que con ello se minimiza el número de almacenamientos intermedios que se pasan entre la rutina y el invocador de la rutina.
Tipos de datos de los parámetros de las rutinas	• Puede mejorar el rendimiento de las rutinas utilizando parámetros VARCHAR en lugar de parámetros CHAR en la definición de las mismas. La utilización de tipos de datos VARCHAR en lugar de tipos de datos CHAR evita que DB2 rellene los parámetros con espacios antes de pasarlos, y reduce el período de tiempo necesario para transmitir el parámetro a través de una red. Por ejemplo, si la aplicación cliente pasa la serie "A SHORT STRING" a una rutina que espera un parámetro CHAR(200), DB2 tiene que rellenar el parámetro con 186 espacios, terminar la serie mediante un nulo y, luego, enviar a la rutina la serie de 200 caracteres entera y el terminador nulo a través de la red. En comparación, pasar la misma serie, *"A SHORT STRING"*, a una rutina que espera un parámetro VARCHAR(200) da como resultado que DB2 pase simplemente la serie de 14 caracteres y un terminador nulo a través de la red.
Inicialización de los parámetros de las rutinas	• Es buena idea inicializar siempre los parámetros de entrada de las rutinas, especialmente si los valores de los parámetros de las rutinas de entrada son nulos. Para los parámetros de las rutinas con un valor nulo, es posible pasar a la rutina un almacenamiento intermedio más corto o vacío en lugar de un almacenamiento intermedio de tamaño completo, con lo que el rendimiento puede mejorar.
Número de variables locales de las rutinas	• Si se minimiza el número de las variables locales declaradas en una rutina, puede mejorarse el rendimiento, al minimizarse el número de sentencias de SQL que se ejecutan en la rutina. • En general, intente utilizar cuantas menos variables mejor. Reutilice las variables si esto no resulta confuso semánticamente.

Tabla 7. Consideraciones sobre el rendimiento y recomendaciones sobre el rendimiento de las rutinas (continuación)

Consideraciones sobre el rendimiento	**Recomendación para el rendimiento**
Inicialización de las variables locales en las rutinas	• Si es posible, resulta aconsejable inicializar varias variables locales de una sola sentencia de SQL ya que con ello se ahorra tiempo en la ejecución total de SQL para la rutina.
Número de conjuntos de resultados que los procedimientos devuelven	• Si puede reducir el número de conjuntos de resultados que una rutina devuelve, mejorará el rendimiento de la rutina.
Tamaño de los conjuntos de resultados que las rutinas devuelven	• Asegúrese de que, para cada conjunto de resultados que devuelva una rutina, la consulta que define el resultado filtre tanto como sea posible las columnas devueltas y el número de filas devueltas. La devolución de columnas o de filas de datos innecesarias no es eficaz y puede hacer que el rendimiento de la rutina no sea óptimo.
Eficacia de la lógica de las rutinas	• Como sucede con cualquier aplicación, el rendimiento de una rutina puede verse limitado por un algoritmo mal implementado. Intente ser lo más eficaz posible al programar las rutinas y siempre que pueda aplique las mejores prácticas de codificación de recomendación general. • Analice su SQL y reduzca la consulta a su forma más simple siempre que pueda. Esto puede conseguirse a menudo utilizando expresiones CASE en vez de sentencias CASE o contrayendo varias sentencias de SQL en una sola sentencia que utilice una expresión CASE como conmutador.

Tabla 7. Consideraciones sobre el rendimiento y recomendaciones sobre el rendimiento de las rutinas (continuación)

Consideraciones sobre el rendimiento	Recomendación para el rendimiento
Modalidad de tiempo de ejecución de la rutina (especificación de la cláusula FENCED o NOT FENCED)	**Uso de la cláusula NOT FENCED:** • En general, es preferible crear la rutina con la cláusula NOT FENCED, que hace que se ejecute en el mismo proceso que el gestor de bases de datos de DB2 en vez de crearla con la cláusula FENCED, que hace que se ejecute en un proceso de DB2 especial fuera del espacio de dirección del motor. • Aunque cabe esperar mejoras en el rendimiento de las rutinas cuando éstas se ejecutan en modalidad sin protección, el código del usuario puede corromper accidental o intencionadamente la base de datos o dañar las estructuras de control de ésta. Sólo debe utilizar la cláusula NOT FENCED cuando necesite maximizar las ventajas sobre el rendimiento y siempre que considera que la rutina es segura. (A fin de informarse acerca de la manera de valorar y mitigar los riesgos de registrar las rutinas C/C++ como NOT FENCED, consulte el tema Seguridad para las rutinas. Si la rutina no es suficientemente segura como para ejecutarla en el proceso del gestor de bases de datos, utilice la cláusula FENCED cuando la cree. Para limitar la creación y ejecución de código potencialmente inseguro, DB2 requiere que un usuario tenga un privilegio especial, CREATE_NOT_FENCED_ROUTINE, a fin de crear rutinas NOT FENCED. • Si se produce una terminación anómala mientras se ejecuta una rutina NOT FENCED, el gestor de bases de datos intentará una recuperación adecuada si la rutina se ha registrado como NO SQL. Sin embargo, el gestor de bases de datos fallará para las rutinas no definidas como NO SQL. • Las rutinas NOT FENCED se deben precompilar con la opción WCHARTYPE NOCONVERT si utilizan datos GRAPHIC o DBCLOB.

Tabla 7. Consideraciones sobre el rendimiento y recomendaciones sobre el rendimiento de las rutinas (continuación)

Consideraciones sobre el rendimiento	Recomendación para el rendimiento
Modalidad de tiempo de ejecución de la rutina (especificación de la cláusula FENCED o NOT FENCED)	**Uso de la cláusula FENCED THREADSAFE** • Las rutinas creadas con la cláusula FENCED THREADSAFE se ejecutan en el mismo proceso que otras rutinas. Más concretamente, las rutinas que no son Java comparten un proceso, mientras que las rutinas Java comparten otro proceso, independiente de las rutinas escritas en otros lenguajes. Esta separación protege a las rutinas Java de las rutinas que potencialmente tienen más tendencia al error escritas en otros lenguajes. Además, el proceso para las rutinas Java contiene una JVM, que provoca un elevado coste de memoria y no se utiliza en los otros tipos de rutinas. Varias invocaciones de rutinas FENCED THREADSAFE generan un compartimiento de recursos que causa una menor actividad general del sistema que las rutinas FENCED NOT THREADSAFE, las cuales se ejecutan, cada una de ellas, en su propio proceso dedicado. • Si piensa que una rutina es suficientemente segura como para ejecutarla en el mismo proceso que otras rutinas, utilice la cláusula THREADSAFE cuando la registre. Al igual que sucede con las rutinas NOT FENCED, encontrará información acerca de la manera de valorar y mitigar los riesgos de registrar las rutinas C/C++ como FENCED THREADSAFE en el tema "Consideraciones sobre seguridad para las rutinas". • Si una rutina FENCED THREADSAFE finaliza de forma anómala, sólo se interrumpe la hebra que está ejecutando esta rutina. Las otras rutinas del proceso siguen en ejecución. Sin embargo, la anomalía que ha ocasionado que esta hebra finalizara anormalmente puede afectar de forma negativa a otras hebras de rutina del proceso y causar que queden retenidas, colgadas o con datos dañados. Después de que una hebra termine anormalmente, se deja de utilizar el proceso en nuevas invocaciones de la rutina. Una vez que todos los usuarios activos finalizan sus trabajos en este proceso, éste se interrumpe. • Cuando se registran rutinas Java, se considera que son THREADSAFE a no ser que se indique lo contrario. Por omisión, todos los otros tipos de LANGUAGE son NOT THREADSAFE. Las rutinas que utilizan LANGUAGE OLE y OLE DB no se pueden especificar como THREADSAFE. • Las rutinas NOT FENCED deben ser THREADSAFE. No es posible registrar una rutina como NOT FENCED NOT THREADSAFE (SQLCODE -104). • Los usuarios de UNIX(R) pueden ver sus procesos THREADSAFE en Java y C buscando db2fmp (Java) o db2fmp (C).

Tabla 7. Consideraciones sobre el rendimiento y recomendaciones sobre el rendimiento de las rutinas (continuación)

Consideraciones sobre el rendimiento	Recomendación para el rendimiento
Modalidad de tiempo de ejecución de la rutina (especificación de la cláusula FENCED o NOT FENCED)	**Modalidad FENCED NOT THREADSAFE** • Las rutinas FENCED NOT THREADSAFE se ejecutan, cada una, en su propio proceso dedicado. Si ejecuta numerosas rutinas, esto puede tener un efecto perjudicial en el rendimiento del sistema de bases de datos. Si la rutina no es suficientemente segura como para ejecutarla en el mismo proceso que otras rutinas, utilice la cláusula NOT THREADSAFE cuando la registre. • En UNIX, los procesos NOT THREADSAFE aparecen como db2fmp (pid) (donde pid es el ID de proceso del agente que utiliza el proceso de modalidad protegida) o como db2fmp (idle) para un db2fmp de tipo NOT THREADSAFE agrupado.
Nivel del acceso a SQL de la rutina: NO SQL, CONTAINS SQL, READS SQL DATA, MODIFIES SQL DATA	• Las rutinas creadas con un nivel bajo de la cláusula de acceso a SQL tendrán un rendimiento mejor que las creadas con un nivel más alto de la cláusula de acceso a SQL. Por lo tanto, deberá declarar las rutinas con el nivel más restrictivo de la cláusula de acceso a SQL. Por ejemplo, si la rutina solamente lee datos de SQL, no la cree con la cláusula MODIFIES SQL DATA sino con la cláusula READS SQL DATA, que es más restrictiva.
Determinismo de la rutina (especificación de la cláusula DETERMINISTIC o NOT DETERMINISTIC)	• La declaración de una rutina con la cláusula DETERMINISTIC o NOT DETERMINISTIC no tiene ningún efecto sobre el rendimiento de la rutina.
Número y complejidad de las acciones externas que la rutina realiza(especificación de la cláusula EXTERNAL ACTION)	• En función del número de acciones externas y de la complejidad de las acciones externas que realice una rutina externa, el rendimiento de la rutina puede verse obstaculizado. Entre los factores que contribuyen a esto se encuentran el tráfico de la red, el acceso de escritura o de lectura a los archivos, el tiempo necesario para ejecutar la acción externa y el riesgo de colgarse asociado con el código o los comportamientos de la acción externa.
Invocación de la rutina cuando los parámetros de entrada son nulos(especificación de la cláusula CALLED ON NULL INPUT)	• Si la recepción de valores nulos en los parámetros de entrada, hace que no se ejecute ninguna lógica y que la rutina devuelva de inmediato, es posible modificar la rutina para que no se invoque completamente cuando se detecten valores nulos en los parámetros de entrada. Para crear una rutina que finalice la invocación de forma prematura si se reciben parámetros de entrada de la rutina, cree la rutina y especifique la cláusula CALLED ON NULL INPUT.

Tabla 7. Consideraciones sobre el rendimiento y recomendaciones sobre el rendimiento de las rutinas (continuación)

Consideraciones sobre el rendimiento	Recomendación para el rendimiento
Parámetros de procedimientos de tipo XML	• Al pasar parámetros del tipo de datos XML, la eficacia es significativamente menor en los procedimientos externos implementados en el lenguaje de programación C o JAVA que en los procedimientos de SQL. Cuando deba pasar uno o más parámetros del tipo de datos XML, plantéese utilizar procedimientos de SQL en vez de procedimientos externos. • Los datos XML se materializan al pasarlos a procedimientos almacenados como parámetros IN, OUT o INOUT. Si utiliza procedimientos almacenados Java, es posible que se tenga que aumentar el tamaño de pila (parámetro de configuración JAVA_HEAP_SZ) de acuerdo con la cantidad y el tamaño de los argumentos XML, así como el número de procedimientos almacenados externos que se estén ejecutando simultáneamente.

Una vez se han creado y desplegado las rutinas, puede resultar más difícil determinar los factores específicos del entorno y de la rutina que están afectando el rendimiento de la rutina, por lo que es importante tener en mente el rendimiento al diseñar las rutinas.

Seguridad de las rutinas

La seguridad de las rutinas es esencial para garantizar su funcionamiento continuado, para minimizar el riesgo de manipulación y para proteger el entorno del sistema de la base de datos. Existen unas cuantas consideraciones sobre las categorías de seguridad de las rutinas, cada una con distintos niveles de riesgo. Se debe ser consciente de estos riesgos al desarrollar o mantener rutinas para mitigar tanto como sea posible los resultados desafortunados.

Control de seguridad de quién puede crear rutinas

La seguridad de las rutinas empieza cuando se otorgan a los usuarios los privilegios necesarios para ejecutar la sentencia CREATE necesaria para crear rutinas en la base de datos. Al otorgar estos privilegios, es importante comprender los riesgos correspondientes:

- Los usuarios con el privilegio para ejecutar la sentencia CREATE sobre una rutina pueden crear varias rutinas.
- Los usuarios con el privilegio para ejecutar la sentencia CREATE sobre una rutina pueden crear rutinas que modifiquen el diseño de la base de datos o los datos de la base de datos en función de qué otros privilegios posea el usuario.
- A los usuarios que creen rutinas satisfactoriamente se les otorgará el privilegio EXECUTE, necesario para invocar la rutina de forma automática.
- A los usuarios que creen rutinas satisfactoriamente se les otorgará el privilegio ALTER ROUTINE, necesario para modificar la rutina.

Para minimizar el riesgo de que los usuarios modifiquen la base de datos y los datos:

- Minimice el número de usuario que poseen el privilegio para crear rutinas.

- Asegúrese de que se eliminen los ID de usuario de los empleados que marchan o, si éstos se reutilizan, asegúrese de valorar los privilegios relativos a los procedimientos.

Consulte los temas sobre cómo controlar el acceso a los datos y los objetos de la base de datos para obtener más información sobre cómo otorgar y revocar privilegios de uno, muchos o todos los usuarios de la base de datos.

Control de seguridad de quién puede invocar rutinas

Resulta sencillo determinar cuándo los usuarios necesitan privilegios: cuando no pueden realizar algo. Resulta más difícil determinar cuándo los usuarios dejan de necesitar estos privilegios. Esto es especialmente cierto cuando se trata de usuarios con privilegios para invocar rutinas, ya que si se les permite que conserven sus privilegios, pueden presentarse riesgos:

- Los usuarios a los que se ha otorgado el privilegio EXECUTE para invocar una rutina continuarán pudiendo invocar la rutina hasta que se elimine dicho privilegio. Si la rutina contiene lógica sensible o actúa sobre datos sensibles, esto puede constituir un riego para la empresa.

Para minimizar el riesgo de que los usuarios modifiquen la base de datos y los datos:

- Minimice el número de usuario que poseen el privilegio para invocar rutinas.
- Asegúrese de que se eliminen los ID de usuario de los empleados que marchan o, si éstos se reutilizan, asegúrese de valorar los privilegios relativos a los procedimientos.
- Si sospecha que alguien está invocando rutinas de forma maliciosa, deberá revocar el privilegio EXECUTE sobre cada una de estas rutinas.

Control de seguridad de las rutinas definidas con cláusulas FENCED o NOT FENCED

Al formular la sentencia CREATE para una rutina, es necesario determinar si se desea especificar la cláusula FENCED o la cláusula NOT FENCED. Una vez comprendidas las ventajas de crear una rutina como protegida o no protegida, es importante valorar los riesgos asociados con la ejecución de rutinas con implementaciones externas como NOT FENCED.

- Las rutinas creadas con la cláusula NOT FENCED pueden corromper accidental o intencionadamente la memoria compartida del gestor de bases de datos, dañar las estructuras de control de las bases de datos o acceder a los recursos del gestor de bases de datos, lo que puede provocar una anomalía en el gestor de bases de datos. También existe el riego de que corrompan las bases de datos y sus tablas.

Para asegurar la integridad del gestor de bases de datos y de sus bases de datos:

- Estudie a fondo las rutinas que se propone crear con la especificación de la cláusula NOT FENCED. Estas rutinas se deben probar y depurar completamente y no pueden mostrar ningún efecto secundario inesperado. Durante el examen del código de la rutina, preste gran atención a la gestión de la memoria y al uso de las variables estáticas. El potencial mayor de corrupción radica cuando el código no gestiona adecuadamente la memoria o cuando utiliza incorrectamente las variables estáticas. Estos problemas son frecuentes en lenguajes distintos de Java(TM) y los lenguajes de programación .NET.

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

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

Seguridad de las rutinas

Al crear las rutinas es importante garantizar que las rutinas, las bibliotecas de rutinas (en el caso de las rutinas externas) y los privilegios de los usuarios que interaccionarán con las rutinas se gestionen teniendo en cuenta la seguridad de la rutina.

Aunque no es necesario tener que elaborar algo tan complejo como una estrategia de seguridad de la rutina, es útil tener en cuenta los factores que contribuyen a la seguridad de las rutinas y seguir un enfoque disciplinado al asegurar las rutinas.

Requisitos previos

- Consulte el tema "Seguridad de las rutinas".
- Para asegurar las rutinas completamente en el sistema de bases de datos, debe disponer de lo siguiente:
  - Acceso de usuario root sobre el sistema operativo del servidor de bases de datos.
  - Autorización DBADM o SYSADM.

Tanto si está creando una rutina como si está valorando una rutina existente, el procedimiento para asegurar una rutina es similar.

1. Limite el número de ID de usuario con los privilegios necesarios para crear rutinas y asegúrese de que estos usuarios tengan permiso para tener estos privilegios.
   - Tras la ejecución satisfactoria de la sentencia CREATE para una rutina, se otorgarán a este ID de usuario de forma automática otros privilegios, incluidos el privilegio EXECUTE, que le permitirá invocar la rutina, y el privilegio GRANT EXECUTE, que le permitirá otorgar a otros usuarios la posibilidad de invocar la rutina.
   - Asegúrese de que los usuarios con este privilegio sean pocos y de que sean los usuarios adecuados los que obtengan este privilegio.
2. Valor la rutina por si el código se ha revisado o probado de forma maligna o inadecuada.
   - Tenga en cuenta el origen de la rutina. ¿La parte que proporciona la rutina resulta fiable?
   - Busque código maligno como, por ejemplo, código que intente leer o escribir en el sistema de archivos del servidor de bases de datos y/o sustituir archivos del mismo.
   - Busque código que se haya implementado incorrectamente relativo con la gestión de la memoria, la manipulación de los punteros y el uso de variables estáticas que podrían causar anomalías en la rutina.
   - Verifique que el código se haya probado de la forma adecuada.

3. Rechace las rutinas que parezca que están codificadas de forma poco segura o incorrecta. El riesgo no siempre vale la pena.
4. Contenga los riesgos asociados con rutinas sólo en parte potencialmente peligrosas.
   - Las rutinas de SQL definidas por el usuario se crean por omisión como rutinas NOT FENCED THREADSAFE porque resulta seguro ejecutarlas en el espacio de memoria del gestor de bases de datos. Para estas rutinas, no es necesario hacer nada.
   - Especifique la cláusula FENCED en la sentencia CREATE de la rutina. Esto garantizará que el funcionamiento de la rutina no afecta al gestor de bases de datos. Ésta es una cláusula por omisión.
   - Si la rutina tiene varias hebras, especifique la cláusula NOT THREADSAFE en la sentencia CREATE de la rutina. Esto garantizará que las anomalías o el código maligno de la rutina no afecten a otras rutinas que se ejecuten en un proceso de hebras compartidas.
5. Si la rutina es una rutina externa, debe poner el archivo de clase o la biblioteca de implementaciones de la rutina en el servidor de bases de datos. Siga las recomendaciones generales para desplegar las rutinas y las recomendaciones específicas para desplegar los archivos de clase o las bibliotecas de rutinas externas.

Autorizaciones y enlace de rutinas que contienen SQL

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

Propietario del paquete

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

Definidor de la rutina

El ID que emite la sentencia CREATE para registrar una rutina. Generalmente, el definidor de la rutina es un DBA, pero, a menudo, también es el propietario del paquete de la rutina. Cuando se invoca una rutina, durante la carga de los paquetes, la autorización para ejecutarla se comprueba con la autorización del definidor para ejecutar el paquete o paquetes asociados con la rutina (no con la autorización del invocador de la rutina). A fin de que una rutina se invoque satisfactoriamente, el definidor de la misma debe disponer del siguiente privilegio o autorización:

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

Si el definidor de la rutina y el propietario del paquete de la rutina son el mismo usuario, el definidor de la rutina tendrá los privilegios EXECUTE

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

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

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

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

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

Como ejemplo, si un paquete asociado con una aplicación que contiene SQL dinámico se ha enlazado con DYNAMICRULES BIND, su ID de autorización de ejecución será el propietario del paquete, no la persona que invoque el paquete. Asimismo, el propietario del paquete será el enlazador real o el valor de la opción de precompilación/enlace OWNER. En este caso, el invocador de la rutina asume este valor en lugar del ID del usuario que está ejecutando la aplicación.

Nota:

1. Para el SQL estático contenido en una rutina, los privilegios del propietario del paquete deben bastar para ejecutar las sentencias de SQL del cuerpo de la rutina. Estas sentencias de SQL pueden requerir privilegios de acceso a tablas o de ejecución si hay referencias anidadas a rutinas.
2. Para el SQL dinámico contenido en una rutina, los privilegios del ID de usuario que se validarán se gobiernan mediante la opción DYNAMICRULES de BIND del cuerpo de la rutina.

3. El propietario del paquete de la rutina debe tener el privilegio GRANT EXECUTE sobre el paquete para el definidor de la rutina. Esto se puede efectuar antes o después de que se registre la rutina, pero es necesario antes de que ésta se invoque, pues, de lo contrario, se devolverá un error (SQLSTATE 42051).

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

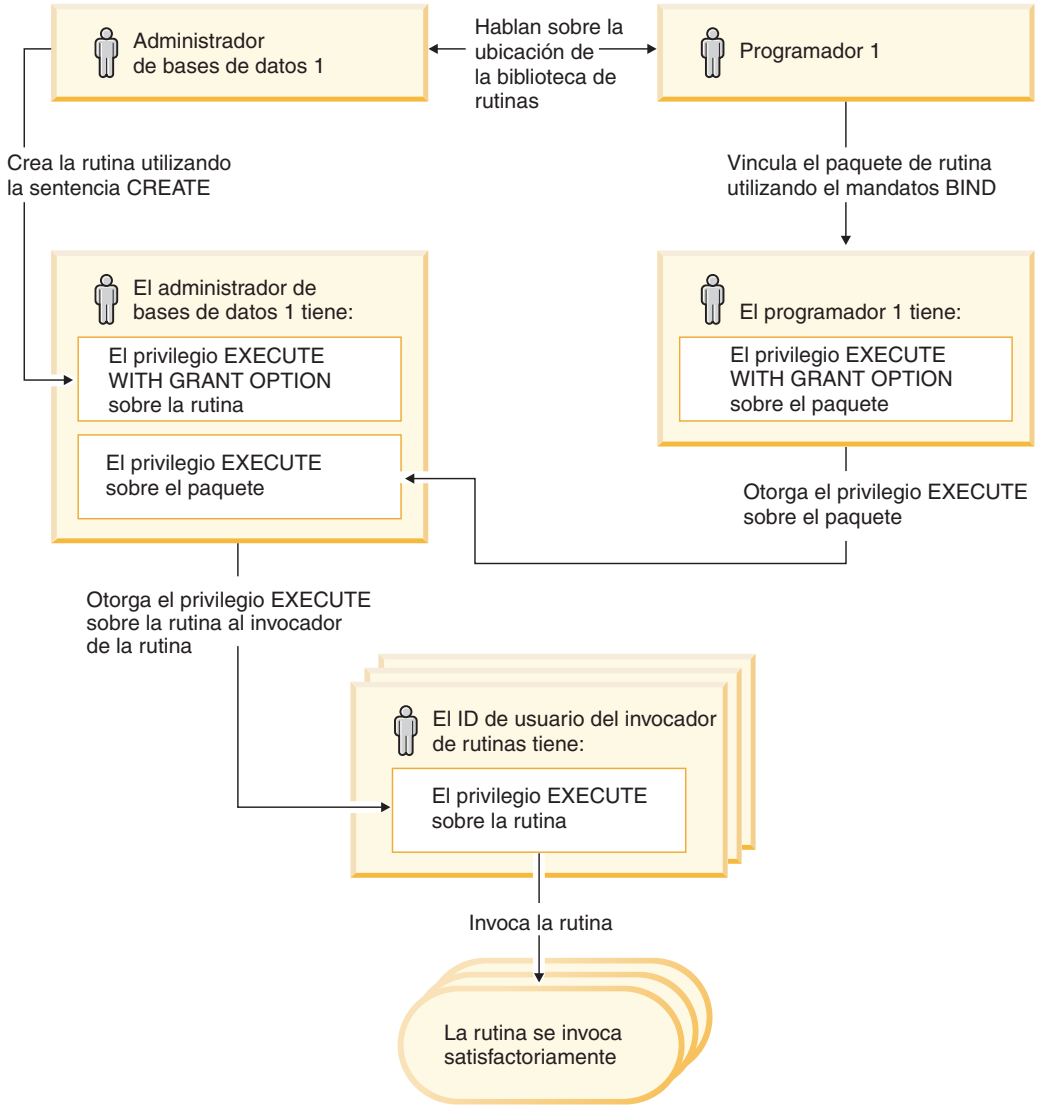


Figura 2. Gestión del privilegio EXECUTE sobre las rutinas

1. El definidor ejecuta la sentencia CREATE adecuada para registrar la rutina. Esta acción registra la rutina en DB2 con su nivel deseado de acceso de SQL, establece la signatura de la rutina y también apunta al ejecutable de la rutina. El definidor, si no es también el propietario del paquete, se tiene que comunicar con los propietarios de paquetes y autores de los programas de rutinas para que quede claro dónde residen las bibliotecas de rutinas, a fin de que se pueda especificar correctamente en la cláusula EXTERNAL de la sentencia CREATE. En virtud de una sentencia CREATE satisfactoria, el definidor tiene el privilegio

EXECUTE WITH GRANT sobre la rutina, pero todavía no tiene el privilegio EXECUTE sobre los paquetes de la rutina.

2. El definidor debe otorgar el privilegio EXECUTE sobre la rutina a los usuarios que tengan permitido utilizar la rutina. (Si el paquete para esta rutina la va a llamar de forma repetitiva, se tiene que realizar este paso antes que el siguiente.)
3. Los propietarios de paquetes precompilan y enlazan el programa de rutina o hacen que otros realicen estas acciones de su parte. Después de una precompilación y enlace satisfactorios, se otorga implícitamente al propietario del paquete el privilegio EXECUTE WITH GRANT OPTION sobre el paquete respectivo. Este paso sigue al primer paso de esta lista, únicamente para abarcar la posibilidad de una repetición de SQL en la rutina. Si tal repetición no existe en ningún caso en particular, la precompilación/enlace puede preceder a la emisión de la sentencia CREATE para la rutina.
4. Cada propietario de un paquete debe otorgar explícitamente al definidor de la rutina el privilegio EXECUTE sobre el paquete de rutina respectivo. Este paso se tiene que llevar a cabo en algún momento posterior al paso anterior. Si el propietario del paquete también es el definidor de la rutina, este paso se puede omitir.
5. Uso estático de la rutina: al propietario de enlace del paquete que hace referencia a la rutina se le tiene que haber asignado el privilegio EXECUTE sobre la rutina, por lo que en este punto se tiene que haber efectuado el paso anterior. Cuando se ejecuta la rutina, DB2 verifica que el definidor tiene el privilegio EXECUTE sobre los paquetes que sean necesarios, por lo que se debe llevar a cabo el paso 3 para cada uno de dichos paquetes.
6. Uso dinámico de la rutina: el ID de autorización, controlado mediante la opción DYNAMICRULES, para la aplicación que realiza la invocación debe tener el privilegio EXECUTE sobre la rutina (paso 4), y el definidor de la rutina debe tener el privilegio EXECUTE sobre los paquetes (paso 3).

Conflictos de datos cuando se leen o se graban procedimientos en tablas

Para conservar la integridad de la base de datos, es necesario evitar conflictos al leer y grabar en tablas. Por ejemplo, suponga que una aplicación está actualizando la tabla EMPLOYEE y que la sentencia llama a una rutina. Suponga que la rutina intenta leer la tabla EMPLOYEE y encuentra la fila que la aplicación está actualizando. La fila tiene un estado indeterminado desde la perspectiva de la rutina: quizá algunas columnas de la fila se han actualizado y otras no. Si la rutina actúa sobre esta fila parcialmente actualizada, puede emprender acciones incorrectas. Para evitar esta clase de problemas, DB2 no permite operaciones que entren en conflicto en ninguna tabla.

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

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

contexto de acceso a una tabla

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

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

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

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

Se aplican las normas siguientes:

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

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

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

Suponga que una aplicación emite la sentencia siguiente:

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

La UDF1 contiene las sentencias siguientes:

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

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

La situación siguiente no producirá un conflicto:

Suponga que una aplicación emite las sentencias siguientes:

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

La UDF2 contiene las sentencias siguientes:

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

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

Capítulo 2. Desarrollo de las rutinas

El desarrollo de rutinas se hace con frecuencia cuando no hay ninguna rutina definida por el sistema disponible que proporcione la funcionalidad requerida.

Aunque hay tipos funcionales de rutinas e implementaciones de rutinas diferentes, los pasos básicos para el desarrollo de rutinas son normalmente comunes para todas las rutinas. Debe determinar el tipo de rutina que ha de crearse, definir la interfaz para la rutina, desarrollar la lógica de la rutina, ejecutar el SQL para crear la rutina, probar la rutina y después desplegarla para su utilización general.

En función del tipo de rutina que opte por desarrollar, existen procedimientos específicos que deberá seguir. Este tema lo dirigirá a los temas adecuados para iniciarse con el desarrollo de las rutinas.

Requisitos previos

- Leer y entender los conceptos básicos de las rutinas:
  - Para aprender, entre otras cosas, acerca de los tipos de rutinas, las aplicaciones útiles para las rutinas, las herramientas para el desarrollo de rutinas y los procedimientos recomendados para las rutinas, consulte el tema:
    - Capítulo 1, "Visión general de las rutinas", en la página 1
- Aprender sobre las herramientas disponibles para el desarrollo de rutinas que agilizan y simplifican el desarrollo de las rutinas:
  - Para aprender sobre las herramientas disponibles para el desarrollo de rutinas, consulte el tema:
    - "Herramientas para desarrollar rutinas" en la página 37

Procedimiento

1. Determine si alguna rutina definida por el sistema existente ya satisface las necesidades de su rutina.
   - Si una rutina definida por el sistema satisface sus necesidades, tal vez prefiera consultar el tema Capítulo 11, "Invocación de rutinas", en la página 355.
2. Determine el tipo funcional de rutina que desea desarrollar.
3. Determine la implementación de la rutina que va a utilizar.
   - Si se necesita una rutina de SQL, consulte la información sobre "Rutinas de SQL" en la página 65.
   - Si se necesita una rutina externa, consulte la información sobre "Rutinas externas" en la página 119.

El desarrollo de rutinas de SQL y de rutinas externas es similar pero existen diferencias. Para ambos tipos de rutinas, primero debe designar la lógica y a continuación crear la rutina en la base de datos. Debe ejecutar una sentencia CREATE específica para el tipo funcional de rutina. Estas sentencias de creación de rutinas incluyen CREATE PROCEDURE, CREATE FUNCTION y CREATE METHOD. Las cláusulas específicas de cada una de las sentencias CREATE definen las características de la rutina, incluido el nombre de la rutina, el número y tipo de parámetros de la rutina y detalles sobre la lógica de la misma. DB2 utiliza la información proporcionada por las cláusulas para identificar y ejecutar la rutina cuando ésta se invoca. Una vez ejecutada satisfactoriamente la sentencia CREATE

© Copyright IBM Corp. 1993, 2010

para una rutina, ésta se crea en la base de datos. Las características de la rutina se almacenan en las tablas catálogos del sistema DB2 que los usuarios pueden consultar. La ejecución de la sentencia CREATE para crear una rutina también se conoce como definición de una rutina o registro de una rutina.

Como en las rutinas externas la lógica se implementa en bibliotecas creadas por el usuario o en las clases situadas en el sistema de archivos de base de datos, se requieren pasos adicionales para programar la lógica, crearla y localizar correctamente el archivo de clase o la biblioteca resultante.

Una vez haya desarrollado las rutinas, es posible que desee:

- Depurar las rutinas
- Desplegar las rutinas en entornos de producción
- Otorgar a los usuarios privilegios para ejecutar las rutinas
- Invocar las rutinas
- Ajustar el rendimiento de las rutinas

Capítulo 3. Visión general de las rutinas SQL

Las rutinas SQL son rutinas cuya lógica se ha implementado con sentencias de SQL exclusivamente, incluidas las sentencias de Lenguaje de procedimientos de SQL (SQL PL). Se caracterizan porque la lógica del cuerpo de la rutina está contenida en la sentencia CREATE que se utiliza para crearlas. Es posible crear procedimientos de SQL, funciones de SQL y métodos de SQL. Aunque todos ellos se implementan en SQL, cada tipo funcional de rutina tiene características distintas.

Antes de decidir la implementación de una rutina de SQL, es importante que primero comprenda qué son las rutinas SQL, cómo se implementan y se utilizan leyendo el tema "Visión general de las rutinas". Una vez comprenda esto, podrá obtener más información acerca de las rutinas de SQL en los temas sobre los conceptos siguientes para tomar decisiones informadas sobre cuándo y cómo utilizarlas en el entorno de bases de datos:

- Procedimientos de SQL
- Funciones de SQL
- Herramientas para el desarrollo de rutinas SQL
- Lenguaje de procedimientos de SQL (SQL PL)
- Comparación entre SQL PL y SQL PL incorporado
- Sentencias y características SQL PL
- Sentencias y características SQL PL incorporado soportadas
- Determinar cuándo utilizar procedimientos o funciones de SQL
- Restricciones de rutinas de SQL

Una vez haya aprendido sobre las rutinas SQL, es posible que desee realizar una de las tareas siguientes:

- Desarrollar procedimientos de SQL
- Desarrollar funciones de SQL
- Desarrollar métodos de SQL

Rutinas de SQL

Las rutinas SQL son rutinas cuya lógica se ha implementado con sentencias de SQL exclusivamente, incluidas las sentencias de Lenguaje de procedimientos de SQL (SQL PL). Se caracterizan porque la lógica del cuerpo de la rutina está contenida en la sentencia CREATE que se utiliza para crearlas. Esto contrasta con las rutinas externas cuya lógica de rutina se implementa en código fuente de programación de formulario construido en biblioteca. Por lo general, las rutinas de SQL pueden contener y ejecutar menos sentencias de SQL que las rutinas externas; no obstante, pueden ser igual de potentes y ofrecer el mismo alto rendimiento al implementarse según los procedimientos recomendados.

Es posible crear procedimientos de SQL, funciones de SQL y métodos de SQL. Aunque todos ellos se implementan en SQL, cada tipo funcional de rutina tiene características distintas.

© Copyright IBM Corp. 1993, 2010

Herramientas para el desarrollo de rutinas SQL

Las herramientas para el desarrollo de rutinas de SQL agilizan y facilitan la creación de procedimiento de SQL. Pueden utilizarse la siguiente GUI y herramientas de línea de mandatos para crear procedimientos de SQL:

- Herramientas de DB2
  - Procesador de línea de mandatos (CLP)
  - IBM Data Studio
  - IBM Database Add-Ins para Microsoft Visual Studio
- El grupo de software de IBM y las herramientas de gestión de la información de IBM
  - IBM Rational Application Developer
  - IBM WebSphere Application Server

Desarrollo de rutinas SQL en IBM Data Studio

La IBM Data Studio es un entorno de desarrollo instalado por separado que proporciona soporte para el desarrollo de rutinas de SQL. Dentro de Data Studio encontrará soporte de herramientas y documentación para realizar las siguientes tareas de desarrollo de rutinas de SQL:

- Crear rutinas de SQL
- Crear rutinas SQL que funcionarán en servidores de bases de datos DB2 para z/OS
- Modificar rutinas de SQL
- Depurar procedimientos de SQL
- Comparar definiciones de rutinas SQL dentro de un proyecto o en un servidor de bases de datos
- Desplegar rutinas de SQL en servidores de bases de datos DB2

También puede hacen muchas más cosas con Data Studio que pueden ayudarle a desarrollar rutinas de SQL, entre ellas: consultar, modificar, cargar y extraer datos de tablas, trabajar con funciones XML, desarrollar rutinas de Java, etc.

Lenguaje de procedimientos de SQL (SQL PL)

El lenguaje de procedimientos de SQL (SQL PL) es un conjunto de sentencias de SQL de DB2 Universal Database versión 7 con el fin de proporcionar las construcciones de procedimientos necesarias para implementar la lógica de flujo de control en consultas y operaciones normales de SQL. SQL PL ha evolucionado desde entonces y el conjunto actual de sentencias de SQL PL y características del lenguaje proporciona soporte para llevar a cabo tareas de programación de alto nivel íntegramente en SQL.

SQL PL es un subconjunto del lenguaje estándar SQL Persistent Stored Modules (SQL/PSM). La especificación del estándar SQL/PSM actual puede encontrarse en ANSI/ISO/IEC 9075-4:1999 Information Technology, Database Language SQL, Part 4: Persistent Stored Modules (SQL/PSM).

SQL PL tiene una sintaxis sencilla que incluye soporte para variables, sentencias condicionales, sentencias en bucle, sentencias de transferencia de control, sentencias de gestión de errores y sentencias de manipulación de conjuntos de resultados. SQL PL puede utilizarse en distintos contextos, como por ejemplo en rutinas con

implementaciones de SQL, y un subconjunto de sentencias de SQL PL puede utilizarse en activadores y sentencias de SQL compuesto dinámico.

SQL PL consta de las sentencias siguientes:

- Sentencias relacionadas con variables
  - DECLARE <variable> DEFAULT <valor>
  - DECLARE <condición>
  - DECLARE <manejador de condiciones>
  - DECLARE CURSOR
  - SET (sentencia-asignación)
- Sentencias condicionales
  - CASE (2 formatos)
  - IF
- Sentencias en bucle
  - FOR
  - LOOP
  - REPEAT
  - WHILE
- Sentencias de transferencia de control
  - CALL
  - GOTO
  - ITERATE
  - LEAVE
  - RETURN
- Sentencias de gestión de errores
  - SIGNAL
  - RESIGNAL
- Sentencias de manipulación de conjuntos de resultados
  - ASSOCIATE LOCATOR(S)
  - ALLOCATE CURSOR

SQL PL incorporado

SQL PL incorporado es un subconjunto de características de SQL PL que pueden utilizarse con sentencias de SQL compuesto dinámico indivisibles que se utilizan en activadores, funciones de SQL, métodos de SQL y código autónomo. El código autónomo hace referencia al lenguaje de scripts SQL PL.

Se describe SQL PL incorporado como "incorporado" porque la lógica se expande y se ejecuta con las sentencias de SQL que hacen referencia a ella.

Las siguientes sentencias de SQL PL están consideradas como parte del conjunto de sentencias de SQL PL incorporado:

- Sentencias relacionadas con variables
  - DECLARE <variable>
  - DECLARE <condition>
  - Sentencia SET (sentencia de asignación)
- Sentencias condicionales

  - IF
  - Expresión CASE
- Sentencias en bucle
  - FOR
  - WHILE
- Sentencias de transferencia de control
  - GOTO
  - ITERATE
  - LEAVE
  - RETURN
- Sentencias de gestión de errores
  - SIGNAL
  - GET DIAGNOSTICS

Otras sentencias de SQL PL que están soportadas en procedimientos de SQL no están soportadas en sentencias compuestas dinámicas. Los cursores y manejadores de condiciones no están soportados en SQL PL incorporado y, por lo tanto, tampoco lo está la sentencia RESIGNAL.

Dado que las sentencias de SQL PL incorporado deben ejecutarse en sentencias compuestas dinámicas, no hay soporte para las sentencias PREPARE, EXECUTE o EXECUTE IMMEDIATE.

Además, al tener que utilizarse en una sentencia de SQL compuesto dinámico indivisible, todas o ninguna de las sentencias de miembro deben comprometerse satisfactoriamente. Por consiguiente, las sentencias COMMIT y ROLLBACK tampoco están soportadas.

En cuanto a las sentencias LOOP y REPEAT, puede utilizarse la sentencia WHILE para implementar la lógica equivalente.

Aunque los activadores y las funciones de SQL solamente dan soporte a las sentencias de SQL PL incorporado, es posible utilizar indirectamente el conjunto más amplio de sentencias de SQL PL, llamando a un procedimiento de SQL desde estas interfaces.

El lenguaje de scripts autónomo con SQL PL incorporado consiste en ejecutar una sentencia compuesta dinámica dentro de un script de procesador de línea de mandatos (CLP) o directamente desde un indicador CLP. Las sentencias de SQL compuesto dinámico están limitadas por las palabras clave BEGIN ATOMIC y END y deben finalizar con un carácter terminador que no sea predeterminado. Pueden contener SQL PL y otras sentencias de SQL.

Dado que las sentencias de SQL PL incorporado se expanden dentro de las sentencias de SQL que les hacen referencia en lugar de compilarse individualmente, deberán tenerse en cuenta ciertas consideraciones relativas al rendimiento cuando planifique si ha de implementar la lógica de procedimiento en SQL PL en un procedimiento de SQL o con SQL PL incorporado en una función, activador o sentencia compuesta dinámica.

SQL PL en procedimientos de SQL

Las sentencias de SQL PL se utilizan fundamentalmente en procedimientos de SQL. Los procedimientos de SQL pueden contener sentencias de SQL básicas para consultar y modificar datos, pero asimismo pueden incluir sentencias de SQL PL para implementar la lógica de flujo de control en las demás sentencias de SQL. El conjunto completo de sentencias de SQL PL puede utilizarse en los procedimientos de SQL.

Los procedimientos de SQL también dan soporte a parámetros, variables, sentencias de asignación, un potente mecanismo de manejo de errores y condiciones, a las llamadas anidadas y repetitivas y a la capacidad de devolución de varios conjuntos de resultados al llamador del procedimiento o a la aplicación cliente.

SQL PL, cuando se utiliza en los procedimientos de SQL, le permite programar efectivamente en SQL. El lenguaje de SQL PL de alto nivel, la escritura intensa de SQL y las características adicionales que proporcionan los procedimientos de SQL hacen que programar con SQL PL resulta rápido y sencillo.

Tome en consideración el ejemplo siguiente como un ejemplo sencillo de las sentencias de SQL PL que están utilizándose en un procedimiento de SQL:

```
CREATE PROCEDURE UPDATE_SAL (IN empNum CHAR(6),
        INOUT rating SMALLINT)
LANGUAGE SQL
BEGIN
  IF rating = 1 THEN
   UPDATE employee
      SET salary = salary * 1.10, bonus = 1000
        WHERE empno = empNum;
  ELSEIF rating = 2 THEN
    UPDATE employee
      SET salary = salary * 1.05, bonus = 500
        WHERE empno = empNum;
  ELSE
    UPDATE employee
      SET salary = salary * 1.03, bonus = 0
        WHERE empno = empNum;
  END IF;
END
```

PL de SQL incorporado y funciones de SQL, activadores y sentencias de SQL compuesto (dinámico)

Las sentencias de SQL PL incorporado pueden ejecutarse desde sentencias de SQL (dinámico), funciones de SQL y activadores.

Una sentencia de SQL (dinámico) compuesto es la que le permite agrupar varias sentencias de SQL en un bloque opcionalmente indivisible, en el cual puede declarar variables y elementos para el manejo de condiciones. Estas sentencias las compila DB2 como una sola sentencia de SQL y pueden contener sentencias PL de SQL incorporadas.

Los cuerpos de las funciones de SQL y activadores pueden contener sentencias de SQL compuesto y también pueden incluir algunas sentencias de SQL PL incorporado.

Por sí solas, las sentencias de SQL compuesto son útiles para crear scripts reducidos que realicen pequeñas unidades de trabajo lógico con un mínimo flujo de control, pero con un flujo de datos significativo. Dentro de funciones y activadores, permiten que se utilice una lógica más compleja al utilizarse esos objetos.

Como ejemplo de sentencia de SQL compuesto dinámico que contiene SQL PL, considere lo siguiente:

```
BEGIN ATOMIC
  FOR row AS
    SELECT pk, c1, discretize(c1) AS v FROM source
  DO
    IF row.v is NULL THEN
      INSERT INTO except VALUES(row.pk, row.c1);
    ELSE
      INSERT INTO target VALUES(row.pk, row.d);
    END IF;
  END FOR;
END
```

La sentencia compuesta está limitada por las palabras clave BEGIN y END. Incluye el uso de las dos sentencias de control FOR e IF/ELSE que forman parte de SQL PL. La sentencia FOR se utiliza para la iteración a través de un conjunto de filas definido. Para cada fila se marca el valor de una columna y, condicionalmente y basándose en el valor, se inserta un conjunto de valores en otra tabla.

Como ejemplo de activador que contiene SQL PL, considere lo siguiente:

```
CREATE TRIGGER validate_sched
NO CASCADE BEFORE INSERT ON c1_sched
FOR EACH ROW
MODE DB2SQL
Vs: BEGIN ATOMIC

  IF (n.ending IS NULL) THEN
    SET n.ending = n.starting + 1 HOUR;
  END IF;

  IF (n.ending > '21:00') THEN
    SIGNAL SQLSTATE '80000'  SET MESSAGE_TEXT =
           'La hora de finalización de la clase es después de las 9';
  ELSE IF (n.DAY=1 or n.DAY-7) THEN
    SIGNAL SQLSTATE '80001' SET MESSAGE_TEXT =
           'No puede planificarse la clase en fin de semana';
  END IF;
END vs;
```

Este activador se activa al realizarse una inserción en una tabla denominada c1_sched y utiliza SQL PL para buscar y proporcionar una hora de finalización de la clase si no se ha proporcionado una, y para emitir un error si la hora de finalización de la clase es posterior a las 9 o si se ha planificado la clase en un fin de semana. Como ejemplo de función SQL escalar que contiene SQL PL, considere lo siguiente:

```
CREATE FUNCTION GetPrice (Vendor CHAR(20), Pid INT)
RETURNS  DECIMAL(10,3)
LANGUAGE SQL  MODIFIES SQL
BEGIN
  DECLARE price DECIMAL(10,3);

  IF Vendor = 'Proveedor 1'
    THEN SET price = (SELECT ProdPrice FROM V1Table WHERE Id = Pid);
  ELSE IF Vendor = 'Proveedor 2'
```

```
    THEN SET price = (SELECT Price FROM V2Table WHERE Pid = GetPrice.Pid);
  END IF;


  RETURN price;
END
```

Esta función simple devuelve un valor de precio escalar, basado en el valor de un parámetro entrado que identifica a un proveedor. También utiliza la sentencia IF.

Si le interesa una lógica más compleja que requiera parámetros de salida, el pase de conjuntos de resultados u otros elementos de procedimiento más avanzados, pueden convenirle más los procedimientos de SQL.

Sentencias CREATE para rutinas de SQL

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

Los procedimientos, funciones y métodos de SQL tienen, cada cual, sus sentencias CREATE respectivas. Aunque la sintaxis de estas sentencias es diferente, existen algunos elementos comunes. En cada una, debe especificar el nombre de la rutina, y los parámetros si debe haber alguno, así como un tipo de retorno. Además, puede especificar palabras clave adicionales que proporcionen información a DB2 sobre la lógica incluida en la rutina. DB2 utiliza el prototipo de rutina y las palabras clave adicionales para identificar la rutina durante la invocación y para ejecutarla con el soporte de las características necesarias y el mejor rendimiento posible.

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

Determinar cuándo deben utilizarse rutinas de SQL o rutinas externas

Al implementar la lógica de rutinas, puede elegir implementar rutinas de SQL o rutinas externas. Hay motivos para elegir cada una de estas dos implementaciones.

Para determinar cuándo debe elegirse implementar una rutina de SQL o una rutina externa, lea lo siguiente para determinar qué factores, si los hay, podrían limitar sus opciones.

- Elija implementar rutinas de SQL si:
  - Las sentencias de SQL PL y SQL proporcionan el soporte adecuado para implementar la lógica que necesita.
  - La lógica de rutinas consta principalmente de sentencias de SQL que consultan o modifican datos, siendo el rendimiento una preocupación. La lógica que contiene una cantidad relativamente pequeña de lógica de flujo de control relativa al número de sentencias de SQL que consultan o modifican datos de la base de datos, generalmente rendirá más con una implementación de rutina de SQL. SQL PL está pensado para ser utilizado para implementar

lógica de procedimiento en las operaciones de base de datos y no principalmente para programa lógica compleja.
  - Las sentencias de SQL que necesita ejecutar pueden ejecutarse en una implementación de rutina externa.
  - Desea hacer que los módulos sean altamente portables entre entornos de sistemas operativos y minimizar la dependencia sobre compiladores de código de lenguaje de programación e intérpretes de scripts.
  - Desea implementar la lógica de forma rápida y fácil utilizando un lenguaje de programación de alto nivel.
  - Se siente más cómodo trabajando con SQL que con lenguajes de programación o de scripts.
  - Desea asegurar la lógica dentro del sistema de gestión de bases de datos.
  - Desea reducir al mínimo el mantenimiento de rutinas y el mantenimiento de paquetes de rutinas al realizar la migración del release o actualizaciones del sistema operativo.
  - Desea reducir al mínimo la cantidad de código necesario para implementar la lógica.
  - Desea utilizar tipos de datos sólidos y reducir al mínimo el riesgo de errores de conversión de tipos de datos.
  - Desea aumentar al máximo la seguridad del código implementado reduciendo al mínimo el riesgo de gestión de memoria, manipulación de punteros u otros peligros de programación comunes.
  - Desea beneficiarse de soporte de antememoria de SQL especial disponible cuando se utiliza PL de SQL.
- Elija implementar un procedimiento externo si:
  - Si la lógica de rutinas es muy compleja y consta de pocas sentencias de SQL, y el rendimiento de las rutinas supone una preocupación. La lógica tal como un algoritmo matemático complejo, que implica una gran cantidad de manipulación de series, o que no accede a la base de datos, generalmente funciona mejor con una implementación de rutina externa.
  - Si las sentencias de SQL que necesita ejecutar pueden ejecutarse en una implementación de rutina externa.
  - La lógica de rutinas realizará llamadas al sistema operativo; esto solamente puede realizarse con rutinas externas.
  - La lógica de rutinas debe leer o escribir en archivos; esto solamente puede realizarse con rutinas externas.
  - Escribir en el sistema de archivos del servidor. Realice esta operación siempre con cuidado.
  - Invocar una aplicación o script que resida en el servidor de bases de datos.
  - Emitir sentencias de SQL concretas que no están soportadas en procedimientos de SQL.
  - Se siente más cómodo programando en un lenguaje de programación que no sea SQL PL.

Por omisión, utilice rutinas SQL si pueden satisfacer sus necesidades. Por lo general es un requisito para implementar lógica compleja o para acceder a archivos o scripts del servidor de bases de datos lo que motiva la decisión de utilizar rutinas externas. Especialmente dado que SQL PL es de aprendizaje e implementación rápido y fácil.

Determinar cuándo utilizar procedimientos o funciones de SQL

Cuando se le plantea la opción de implementar lógica con SQL PL en un procedimiento de SQL o una función de SQL, hay motivos para elegir cada una de estas dos implementaciones.

Lea lo siguiente para determinar cuándo elegir el uso de un procedimiento de SQL o una función de SQL.
Elija implementar una función de SQL si:

- Una función de SQL puede satisfacer los requisitos funcional y no espera necesitar más adelante las características que proporciona un procedimiento de SQL.
- El rendimiento es una prioridad y la lógica que debe contenerse en la rutina consta solamente de consultas o devuelve solamente un único conjunto de resultados.

  Cuando solamente contienen consultas o la devolución de un solo conjunto de resultados una función de SQL funciona mejor que un procedimiento de SQL equivalente lógicamente, debido a cómo se compilan las funciones de SQL.

  En los procedimientos de SQL, las consultas estáticas en forma de sentencias SELECT y sentencias full-select se compilan individualmente, de forma que cada consulta se convierte en una sección de un plan de acceso a consultas en un paquete cuando se crea el procedimiento de SQL. No hay una recompilación de este paquete hasta que vuelve a crearse el procedimiento de SQL o el paquete vuelve a la base de datos. Esto significa que el rendimiento de las consultas se determina basándose en la información disponible para el gestor de bases de datos en un momento anterior al de ejecución del procedimiento de SQL y, por consiguiente, podría no resultar óptimo. Además, con un procedimiento de SQL también está implicada una actividad adicional cuando el gestor de bases de datos cambia de ejecutar sentencias de flujo de procedimientos a sentencias de SQL que consultan o modifican datos.

  Las funciones de SQL, no obstante, se expanden y compilan dentro de la sentencia de SQL que les hace referencia, lo que significa que se compilan cada vez que se compila la sentencia de SQL lo que, dependiendo de la sentencia, podría suceder de forma dinámica. Dado que las funciones de SQL no están asociadas directamente con un paquete, no hay una actividad adicional cuando el gestor de bases de datos cambie de ejecutar sentencias de flujo de procedimientos a sentencias de SQL que consultan o modifican datos.

Elija implementar un procedimiento de SQL si:

- Son necesarias características de SQL PL que solamente están soportadas en procedimientos de SQL. Esto incluye: soporte de parámetros de salida, uso de un cursor, la capacidad de devolver varios conjuntos de resultados al llamador, soporte completo de manejo de condiciones, control de transacciones y punto de salvar.
- Desea ejecutar sentencias no de SQL PL que solamente pueden ejecutarse en procedimientos de SQL.
- Desea modificar datos y la modificación de datos no está soportada para el tipo de función que necesita.

Aunque no siempre es obvio, a menudo se pueden reescribir fácilmente los procedimientos de SQL como funciones de SQL que realicen una lógica equivalente. Esta puede ser una manera eficaz de aumentar al máximo el rendimiento cuando cada pequeña mejora del rendimiento cuenta.

Determinar cuándo utilizar sentencias de SQL compuesto dinámico o procedimientos de SQL

Al determinar cómo implementar un bloque atómico de SQL PL y otras sentencias de SQL, puede que tenga que elegir entre la utilización de sentencias de SQL compuesto dinámico o procedimientos de SQL. Aunque los procedimientos de SQL utilizan internamente sentencias de SQL compuesto dinámico, la opción de utilizar unos u otras puede depender de otros factores.

Rendimiento

Si una sentencia de SQL compuesto dinámico puede satisfacer sus necesidades funcionales, es preferible utilizarla, ya que las sentencias de SQL que son miembros de una sentencia de SQL compuesto dinámico se compilan y ejecutan como un solo bloque. Además, estas sentencias funcionan generalmente mejor que las sentencias CALL para los procedimientos de SQL lógicamente equivalentes.

Durante la creación del procedimiento de SQL, se compila el procedimiento y se crea un paquete. El paquete contiene la mejor vía de ejecución para acceder a los datos a partir del tiempo de compilación del procedimiento de SQL. Las sentencias de SQL compuesto dinámico se compilan cuando se ejecutan. La mejor vía de ejecución para acceder a los datos de estas sentencias se determina utilizando la información de base de datos más actualizada, que puede indicar que su plan de acceso puede ser mejor que el de un procedimiento de SQL lógicamente equivalente creado en un momento anterior, lo que significa a su vez que puede funcionar mejor.

Complejidad de la lógica necesaria

Si la lógica que debe implementarse es compleja, probablemente serán preferibles los procedimientos de SQL, ya que dan soporte al SQL PL exhaustivo con el que puede implementar fácilmente una lógica eficiente.

Si la lógica es lo bastante simple y el número de sentencias es relativamente pequeño, considere la posibilidad de utilizar SQL PL incorporado en una función o sentencia o compuesta dinámica. Los procedimientos de SQL pueden manejar lógica simple, pero la utilización de procedimientos de SQL representa un exceso de actividad general que es mejor evitar si no son necesarios.

Número de sentencias de SQL que debe ejecutarse

En casos en los que sólo deban ejecutarse una o dos sentencias de SQL, es posible que la utilización de un procedimiento de SQL no represente ninguna ventaja. En realidad, podría afectar negativamente al rendimiento total necesario para ejecutar estas sentencias. En tales casos, es mejor utilizar PL SQL incorporado en una sentencia compuesta dinámica.

Atomicidad y control de transacción

La atomicidad es otro factor a tener en cuenta. Las sentencias compuestas dinámicas deben ser atómicas. Los compromisos y retrotracciones no están soportados en sentencias compuestas dinámicas. Si es necesario el control de transacción o el soporte de retrotracción hasta un punto de salvación, deben utilizarse procedimientos de SQL.

Seguridad

La seguridad también puede ser un factor a tener en cuenta. Sólo los usuarios con el privilegio EXECUTE sobre el procedimiento pueden ejecutar procedimientos de SQL. Esto puede ser de utilidad si necesita limitar quién puede ejecutar una parte determinada de la lógica. La capacidad de ejecutar una sentencia compuesta dinámica también puede gestionarse; sin embargo, la autorización de ejecución de procedimiento de SQL proporciona un nivel adicional de control de la seguridad.

Soporte de características

Si desea aprovechar muchas de las características sólo disponibles en procedimientos de SQL, incluidos el uso de parámetros, manejadores de condiciones y errores y la posibilidad de devolver uno o varios conjuntos de resultados al llamador, debe utilizar procedimientos de SQL.

Modularidad. longevidad y reutilización

Los procedimientos de SQL son objetos de base de datos que se almacenan de forma persistente en la base de datos a los que varias aplicaciones o scripts pueden hacer referencia de forma coherente. Las sentencias compuestas dinámicas no se almacenan en la base de datos y, por tanto, la lógica que contienen no puede reutilizarse fácilmente.

Por omisión, utilice rutinas SQL si pueden satisfacer sus necesidades. Generalmente, es la necesidad de implementar una lógica compleja o utilizar las características soportadas por procedimientos de SQL, que no están disponibles en sentencias compuestas dinámicas lo que motiva la decisión de utilizar procedimientos de SQL.

Reescritura de procedimientos de SQL como funciones definidas por el usuario de SQL

Para aumentar al máximo el rendimiento en un sistema de gestión de bases de datos, si es posible, a veces puede resultar beneficioso reescribir procedimientos de SQL simples como funciones de SQL. Los procedimientos y las funciones comparten el hecho de que los cuerpos de las rutinas se implementan con un bloque compuesto que puede contener SQL PL. En ambos casos, se incluyen las mismas sentencias SQL PL dentro de bloques compuestos delimitados por las palabras claves BEGIN y END.

Deben tenerse en cuenta algunos puntos al convertir un procedimiento de SQL a una función de SQL:

- La principal y única razón para hacerlo es mejorar el rendimiento de las rutinas cuando la lógica solamente consulta datos
- En una función escalar podría tener que declarar variables para mantener el valor de retorno para sortear el no poder asignar directamente un valor a un parámetro de salida de la función. El valor de salida es una función escalar definida por el usuario solamente se especifica en la sentencia RETURN para la función.
- Si una función de SQL va a modificar datos, debe crearse explícitamente utilizando la cláusula de SQL MODIFIES para que pueda contener sentencias de SQL que modifican datos.

En el ejemplo siguiente se muestra un procedimiento de SQL y una función escalar de SQL equivalentes lógicamente. Estas dos rutinas proporcionan funcionalmente el mismo valor de salida dados los mismos valores de entrada; sin embargo, se implementan y se invocan de maneras ligeramente diferentes.

```
CREATE PROCEDURE GetPrice (IN Vendor CHAR(20),
                           IN Pid INT,
                           OUT price DECIMAL(10,3))
LANGUAGE SQL
BEGIN

  IF Vendor = 'Proveedor 1'
   THEN SET price = (SELECT ProdPrice FROM V1Table WHERE Id = Pid);
  ELSE IF Vendor = 'Proveedor 2'
   THEN SET price = (SELECT Price FROM V2Table
                  WHERE Pid = GetPrice.Pid);
  END IF;
  END
```

Este procedimiento toma dos valores de parámetro de entrada y devuelve un valor de parámetro de salida que se determina condicionalmente basándose en los valores del parámetro de entrada. Utiliza la sentencia IF. Se llama a este procedimiento de SQL ejecutando la sentencia CALL. Por ejemplo, desde el CLP podría ejecutar lo siguiente:

```
CALL GetPrice( 'Proveedor 1',  9456, ?)
```

El procedimiento de SQL puede reescribirse como una función de tabla de SQL equivalente lógicamente, de la siguiente manera:

```
CREATE FUNCTION GetPrice (Vendor CHAR(20), Pid INT)
    RETURNS  DECIMAL(10,3)
LANGUAGE SQL  MODIFIES SQL
BEGIN
  DECLARE price DECIMAL(10,3);

  IF Vendor = 'Proveedor 1'
    THEN SET price = (SELECT ProdPrice FROM V1Table WHERE Id = Pid);
  ELSE IF Vendor = 'Proveedor 2'
    THEN SET price = (SELECT Price FROM V2Table
              WHERE Pid = GetPrice.Pid);
  END IF;


  RETURN price;
END
```

Esta función toma dos parámetros de entrada y devuelve un solo valor escalar, basado condicionalmente en los valores del parámetro de entrada. Requiere la declaración y uso de una variable local denominada 'price' para contener el valor a devolver hasta que vuelva la función, mientras que el procedimiento de SQL puede utilizar el parámetro de salida como una variable. Funcionalmente, estas dos rutinas utilizan la misma lógica.

Por supuesto, ahora la interfaz de ejecución para cada una de estas rutinas es distinta. En lugar de simplemente llamar al procedimiento de SQL con la sentencia CALL, la función de SQL debe invocarse dentro de una sentencia de SQL donde se permita una expresión. En la mayoría de casos, esto no resulta un problema y, de hecho, podría ser beneficioso si la intención es operar de inmediato sobre los datos devueltos por la rutina. Estos son dos ejemplos de cómo puede invocarse la función de SQL.

Puede invocarse utilizando la sentencia VALUES:

```
VALUES (GetPrice('Proveedor 1', 9456))
```

También puede invocarse en una sentencia SELECT que, por ejemplo, podría seleccionar valores de una tabla y filtrar filas basándose en el resultado de la función:

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

Capítulo 4. Desarrollar procedimientos de SQL

El desarrollo de procedimientos de SQL es similar al desarrollo de otros tipos de rutinas. El desarrollo de procedimientos de SQL comprende todos los pasos necesarios, desde la fase de diseño a la fase de despliegue.

- Procedimientos de SQL
- Diseñar procedimientos de SQL
- Crear procedimientos de SQL
- Llamar a procedimientos de SQL
- Depurar procedimientos de SQL
- Ajustar procedimientos de SQL
- Desplegar procedimientos de SQL

Para facilitar el desarrollo de procedimientos de SQL, puede consultar varios ejemplos de procedimientos de SQL. Estos ejemplos muestran diversos tipos de procedimientos de SQL que, a su vez, muestran algunas de las diversas características de procedimientos de SQL.

Procedimientos de SQL

Los procedimientos de SQL son procedimientos implementados por completo con SQL que sirven para encapsular lógica que puede invocarse como una subrutina de programación. Existen muchas aplicaciones útiles de los procedimientos de SQL en una base de datos o en la arquitectura de una de aplicación de base de datos. Los procedimientos de SQL pueden utilizarse para crear scripts simples destinados a solicitar la transformación y actualización de datos con rapidez o para generar informes básicos, mejorar el rendimiento de las aplicaciones, modularizar las aplicaciones y mejorar el diseño global y la seguridad de la base de datos.

Existen muchas características de los procedimientos de SQL que los convierten en potentes opciones de utilización de rutinas.

Antes de decidir la implementación de un procedimiento de SQL, es importante que comprenda qué son los procedimientos de SQL en el contexto de las rutinas SQL, cómo se implementan y cómo pueden utilizarse, primero aprendiendo sobre las rutinas y luego consultando el tema "Visión general de los procedimientos de SQL".

Características de los procedimientos de SQL

Los procedimientos de SQL se caracterizan por muchas características. Los procedimientos de SQL:

- Pueden contener sentencias de Lenguaje de procedimientos (PL) de SQL y características que den soporte a la implementación de lógica de flujo de control alrededor de sentencias de SQL estático y dinámico tradicionales.
- Están soportados en toda la familia DB2 de productos de base de datos, en la que están soportadas muchas o casi todas las características soportadas en DB2 Versión 9.
- Son fáciles de implementar, ya que utilizan un lenguaje simple altamente tipificado y de alto nivel.

© Copyright IBM Corp. 1993, 2010

- Los procedimientos de SQL son más fiables que los procedimientos externos equivalentes.
- Se ajustan al estándar SQL SQL99 ANSI/ISO/IEC.
- Dan soporte a modalidades de pase de parámetros de entrada, salida y entrada-salida.
- Dan soporte a un modelo de manejo de errores y condiciones sencillo pero potente.
- Permiten devolver varios conjuntos de resultados al llamador o a una aplicación cliente.
- Permiten acceder fácilmente a los valores SQLSTATE y SQLCODE como variables especiales.
- Residen en la base de datos y se copian (copia de seguridad) y restauran automáticamente.
- Pueden invocarse siempre que está soportada la sentencia CALL.
- Dan soporte a llamadas de procedimiento anidado a otros procedimientos de SQL o a procedimientos implementados en otros lenguajes.
- Dan soporte a la recursión.
- Dan soporte a puntos de salvación y a la retrotracción de sentencias de SQL ejecutadas para suministrar control de transacción extensivo.
- Pueden llamarse desde activadores.

Los procedimientos de SQL suministran soporte extensivo ilimitado a las características indicadas anteriormente. Si se implementan según los procedimientos recomendados, pueden jugar un papel esencial en la arquitectura, el diseño de aplicaciones y el rendimiento del sistema de las bases de datos.

Diseño de procedimientos de SQL

El diseño de procedimientos de SQL requiere entender los requisitos, las características de los procedimientos de SQL, cómo utilizar las características de SQL y conocimiento de las restricciones que pudieran dificultar el diseño. Los siguientes temas sobre el diseño de procedimientos de SQL le ayudará a aprender a diseñar procedimientos de SQL que utilicen de forma óptima las características de los procedimientos de SQL.

- Componentes de los procedimientos de SQL
- Consideraciones para procedimientos almacenados de SQL para varias plataformas
- Sentencias de SQL PL y características de lenguaje soportadas en procedimientos de SQL
- Consideraciones sobre OLTP para procedimientos de SQL
- Rendimiento de los procedimientos de SQL
- Reescritura de procedimientos de SQL como funciones definidas por el usuario de SQL
- Manejo de errores y avisos de DB2

Componentes de los procedimientos de SQL

Para entender los procedimientos de SQL resulta de ayuda entender primero las partes de un procedimiento de SQL. Estas son algunas de las partes de los procedimientos de SQL:

- Estructura de los procedimientos de SQL

- Parámetros en los procedimientos de SQL
- Variables en los procedimientos de SQL
- SQLCODE y SQLSTATE en procedimientos de SQL
- Bloques indivisibles y ámbito de las variables en los procedimientos de SQL
- Cursores en los procedimientos de SQL
- Unidades lógicas en SQL PL
- Manejadores de condiciones y errores en los procedimientos de SQL
- Sentencias de SQL que pueden ejecutarse en procedimientos de SQL

Estructura de los procedimientos de SQL

Los procedimientos de SQL constan de varios componentes lógicos, y el desarrollo de SQL requiere que el usuario implemente dichos componentes según un formato estructurado. El formato es bastante directo y fácil de seguir y está destinado a simplificar el diseño y la semántica de las rutinas.

El núcleo de un procedimiento de SQL es una sentencia compuesta. Las sentencias compuestas están limitadas por las palabras clave BEGIN y END. Estas sentencias pueden ser ATOMIC o NOT ATOMIC. Por omisión, son NOT ATOMIC.

Dentro de una sentencia compuesta pueden declararse varios objetos SQL PL opcionales, a los que puede hacerse referencia con sentencias de SQL. El diagrama que sigue ilustra el formato estructurado de una sentencia compuesta dentro de los procedimientos de SQL:

```
etiqueta:  BEGIN
  Declaraciones de variables
  Declaraciones de condiciones
  Declaraciones de cursor
  Declaraciones de manejador de condición
  Sentencias de asignación, flujo de control, sentencias SQL u otras
   sentencias compuestas
END etiqueta
```

El diagrama muestra que los procedimientos de SQL pueden constar de una o varias sentencias compuestas opcionalmente atómicas (o bloques) y que dichos bloques pueden estar anidados o introducidos uno tras otro dentro de un solo procedimiento de SQL. Dentro de cada uno de estos bloques atómicos, existe un orden prescrito para las declaraciones opcionales de variable, condición y manejador. Éstas deben preceder a la introducción de la lógica de procedimiento implementada con sentencias de control SQL y otras sentencias de SQL y declaraciones de cursor. Los cursores pueden declararse en cualquier lugar con el conjunto de sentencias de SQL contenidas en el cuerpo del procedimiento de SQL.

Para clarificar el flujo de control, los bloques atómicos de procedimientos de SQL pueden etiquetarse al igual que muchas de las sentencias de control SQL contenidas en ellos. Esto facilita la exactitud al hacer referencia a variables y la transferencia de referencias de sentencia de control.

A continuación figura un ejemplo de un procedimiento de SQL que presenta cada uno de los elementos listados anteriormente:

```
CREATE PROCEDURE DEL_INV_FOR_PROD (IN prod INT, OUT err_buffer VARCHAR(128))
LANGUAGE SQL
DYNAMIC RESULT SETS 1
BEGIN

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

```
  DECLARE NO_TABLE CONDITION FOR SQLSTATE '42704';
  DECLARE cur1 CURSOR WITH RETURN TO CALLER
                 FOR SELECT * FROM Inv;

  A: BEGIN ATOMIC
       DECLARE EXIT HANDLER FOR NO_TABLE
         BEGIN
            SET ERR_BUFFER='Table Inv does not exist';
         END;


       SET err_buffer = '';

       IF (prod < 200)
         DELETE FROM Inv WHERE product = prod;
       ELSE IF (prod < 400)
         UPDATE Inv SET quantity = 0 WHERE product = prod;
       ELSE
         UPDATE Inv SET quantity = NULL WHERE product = prod;
       END IF;

    B:  OPEN cur1;

END
```

Sentencias compuestas NOT ATOMIC en procedimientos de SQL

El ejemplo anterior ilustraba una sentencia compuesta NOT ATOMIC, que es el tipo predeterminado utilizado en procedimientos de SQL. Si se produce una condición de error no manejada dentro de una sentencia compuesta, los trabajos realizados antes del error no se retrotraerán, pero tampoco se comprometerán. El grupo de sentencias sólo puede retrotraerse si la unidad de trabajo se retrotrae explícitamente mediante sentencias ROLLBACK o ROLLBACK TO SAVEPOINT. También puede utilizar la sentencia COMMIT para comprometer sentencias satisfactorias si procede.

A continuación, figura un ejemplo de procedimiento de SQL con una sentencia compuesta NOT ATOMIC:

```
CREATE PROCEDURE not_atomic_proc ()
LANGUAGE SQL
SPECIFIC not_atomic_proc
 nap:  BEGIN NOT ATOMIC

 INSERT INTO c1_sched (class_code, day)
   VALUES ('R11:TAA', 1);

 SIGNAL SQLSTATE '70000';

 INSERT INTO c1_sched (class_code, day)
   VALUES ('R22:TBB', 1);

END nap
```

Cuando la sentencia SIGNAL se ejecuta, lanza explícitamente un error que no se ha manejado. El procedimiento efectúa el retorno inmediatamente después. Una vez efectuado el retorno, aunque se ha producido un error, la primera sentencia INSERT se ha ejecutado satisfactoriamente y ha insertado una fila en la tabla c1_sched. El procedimiento no ha comprometido ni retrotraído la inserción de fila, y esto queda sin hacer en toda la unidad de trabajo en la que se ha llamado al procedimiento de SQL.

Sentencias compuestas ATOMIC en procedimientos de SQL

Como sugiere su nombre, las sentencias compuestas ATOMIC pueden considerarse como una sola unidad. Si surgen dentro de ellas condiciones de error no manejadas, todas las sentencias que se han ejecutado hasta ese punto se consideran también anómalas, y por tanto se retrotraen.

Las sentencias compuestas atómicas no pueden estar anidadas dentro de otras sentencias compuestas de tipo ATOMIC.

No puede utilizarse la sentencia SAVEPOINT, la sentencia COMMIT ni la sentencia ROLLBACK desde el interior de una sentencia compuesta ATOMIC. Estas sentencias sólo están soportadas en sentencias compuestas NOT ATOMIC dentro de procedimientos de SQL.

A continuación, figura un ejemplo de procedimiento de SQL con una sentencia compuesta ATOMIC:

```
CREATE PROCEDURE atomic_proc ()
LANGUAGE SQL
SPECIFIC atomic_proc

ap:  BEGIN ATOMIC

   INSERT INTO c1_sched (class_code, day)
     VALUES ('R33:TCC', 1);

   SIGNAL SQLSTATE '70000';

   INSERT INTO c1_sched (class_code, day)
     VALUES ('R44:TDD', 1);

END ap
```

Cuando la sentencia SIGNAL se ejecuta, lanza explícitamente un error que no se ha manejado. El procedimiento efectúa el retorno inmediatamente después. La primera sentencia INSERT se retrotrae independientemente de que la ejecución satisfactoria dé como resultado una tabla sin filas insertadas para este procedimiento.

Etiquetas y sentencias compuestas de procedimiento de SQL

Las etiquetas pueden utilizarse opcionalmente para nombrar cualquier sentencia ejecutable de un procedimiento de SQL, incluidas sentencias compuestas y bucles. Haciendo referencia a etiquetas de otras sentencias, puede obligar al flujo de ejecución a saltar desde una sentencia compuesta o bucle, o incluso a saltar al principio de una sentencia compuesta o bucle. Puede hacerse referencia a las etiquetas mediante sentencias GOTO, ITERATE y LEAVE.

Opcionalmente, puede suministrar una etiqueta correspondiente para el final (END) de una sentencia compuesta. Si se suministra una etiqueta de finalización, debe ser la misma que la etiqueta utilizada para el principio.

Cada etiqueta debe ser exclusiva dentro del cuerpo de un procedimiento de SQL.

Las etiquetas también pueden utilizarse para evitar la ambigüedad si se ha declarado una variable con el mismo nombre en más de una sentencia compuesta en el procedimiento almacenado. Puede utilizarse una etiqueta para calificar el nombre de una variable SQL.

Soporte a matrices en procedimientos SQL

Los procedimientos SQL dan soporte a parámetros y variables de tipo de matriz. Las matrices permiten pasar de forma práctica colecciones temporales de datos entre una aplicación y un procedimiento almacenado o entre dos procedimientos almacenados.

Dentro de los procedimientos almacenados de SQL, las matrices se pueden manipular como se hace con las matrices en lenguajes de programación convencionales. Es más, las matrices se integran dentro del modelo relacional de forma que los datos que representan una matriz se pueden convertir con facilidad en una tabla y los datos en una columna de la tabla que se puede añadir a la matriz. En los siguientes ejemplos se muestran varias operaciones que se realizan con matrices. Ambos ejemplos son scripts del procesador de línea de mandatos (CLP) que utilizan el carácter de porcentaje (%) como un terminador de la sentencia.

Ejemplo 1

En este ejemplo se muestran dos procedimientos, sub y main. El procedimiento main crea una matriz de 6 enteros mediante un constructor array. A continuación pasa la matriz al procedimiento sum, que calcula la suma de los elementos de la matriz de entrada y, a continuación, devuelve el resultado a main. El procedimiento sum muestra la forma en que se indexan los elementos de una matriz y la función CARDINALITY, que devuelve el número de elementos en la matriz.

```
create type intArray as integer array[100] %

create procedure sum(in numList intArray, out total integer)
begin
declare i, n integer;

set n = CARDINALITY(numList);

set i = 1;
set total = 0;

while (i <= n) do
set total = total + numList[i];
set i = i + 1;
end while;

end %

create procedure main(out total integer)
begin
declare numList intArray;

set numList = ARRAY[1,2,3,4,5,6];

call sum(numList, total);

end %
```

Ejemplo 2

En este ejemplo, se utilizan dos tipos de datos de matriz (intArray y stringArray), y una tabla persons con dos columnas (id y name). El procedimiento processPersons añade tres personas más a la tabla y devuelve una matriz con los name de persons que contengan la letra 'o' ordenados según el id. Los id y los nombres de las tres personas a añadir se representan como dos matrices (ids y

names). Estas matrices se utilizan como argumentos de la función UNNEST, que convierte las matrices en una tabla de dos columnas, y cuyos elementos se añaden a continuación en la tabla persons. Por último, la última sentencia en el procedimiento utiliza la función agregada ARRAY_AGG para calcular el valor del parámetro de salida.

```
create type intArray as integer array[100] %
create type stringArray as varchar(10) array[100] %

create table persons (id integer, name varchar(10)) %
insert into persons values(2, 'Tom') %
insert into persons values(4, 'Jill') %
insert into persons values(1, 'Joe') %
insert into persons values(3, 'Mary') %

create procedure processPersons(out witho stringArray)
begin
declare ids intArray;
declare names stringArray;

set ids = ARRAY[5,6,7];
set names = ARRAY['Bob', 'Ann', 'Sue'];

insert into persons(id, name)
(select T.i, T.n from UNNEST(ids, names) as T(i, n));

set witho = (select array_agg(name order by id)
from persons
where name like '%o%');
end %
```

Parámetros en los procedimientos de SQL

Los procedimientos de SQL soportan parámetros para pasar valores de SQL a los procedimientos o para extraerlos de ellos.

Los parámetros pueden ser útiles en los procedimientos de SQL al implementar lógica que es condicional en una entrada o conjunto de entradas de valores escalares, o cuando necesita devolver uno o varios valores escalares de salida y no desea devolver un conjunto de resultados.

Es aconsejable entender las características y las limitaciones de los parámetros en los procedimientos de SQL al diseñar o crear procedimientos de SQL.

- DB2 ofrece soporte al uso opcional de un gran número de parámetros de entrada, de salida y de entrada-salida en los procedimientos de SQL. Las palabras clave IN, OUT e INOUT de la parte de signatura de rutina de las sentencias 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 especifican varios parámetros para un procedimiento, cada uno debe tener un nombre exclusivo.
- Si hay que declarar una variable en el procedimiento con el mismo nombre que un parámetro, se debe declarar en un bloque indivisible etiquetado anidado dentro del procedimiento. De lo contrario, DB2 detectará lo que sino sería una referencia de nombre ambiguo.
- Los parámetros de los procedimientos de SQL no pueden recibir el nombre de SQLSTATE ni SQLCODE, independientemente del tipo de datos del parámetro.

Consulte la sentencia CREATE PROCEDURE (SQL) para obtener detalles completos sobre las referencias de parámetros en los procedimientos de SQL.

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

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

Marcadores de parámetros

Un marcador de parámetro, que a menudo se indica con un signo de interrogación (?), es un contenedor en una sentencia cuyo valor se obtiene al ejecutar la sentencia. Una aplicación asocia marcadores de parámetros a variables de la aplicación. Durante la ejecución de la sentencia, los valores de estas variables sustituyen el correspondiente marcador de parámetro. Podría darse una conversión de datos cuando se realiza este proceso.

Ventajas que ofrecen los marcadores de parámetros

En el caso de sentencias SQL que se deben ejecutar muchas veces, muchas veces es beneficioso preparar la sentencia SQL una única vez,y volver a utilizar el plan de la consulta utilizando marcadores de parámetros para sustituir los valores de entrada en tiempo de ejecución. In DB2® 9, un marcador de parámetro se representa con uno de estos dos formatos:

- En el primer formato, mediante un carácter "?", se utiliza en la ejecución SQL dinámica (SQL incorporado dinámico, CLI, Perl, etc).
- El segundo formato corresponde a una construcción estándar de SQL incorporado en el que el nombre de la variable se indica mediante un carácter de dos puntos (:) como prefijo (:variable1). Este formato se utiliza en la ejecución de SQL estático y a menudo se hace referencia a él como una variable de sistema principal.

Cualquiera de estos dos formatos indica cuándo debe sustituirse una variable de aplicación dentro de una sentencia SQL. Se hace referencia a los marcadores de parámetro mediante números, ordenados de forma secuencial de izquierda a derecha, empezando por el uno. Antes de que se ejecute la sentencia SQL, la aplicación debe vincular un área de almacenamiento de variable con cada marcador de parámetro que se especifica en la sentencia SQL. Además las variables vinculadas deben ser un área de almacenamiento válida y deben contener valores de datos de entrada cuando se ejecute la sentencia preparada en relación a la base de datos.

En el siguiente ejemplo se muestra una sentencia SQL con dos marcadores de parámetro.

```
SELECT * FROM customers WHERE custid = ? AND lastname = ?
```

Tipos soportados

DB2 da soporte a marcadores de parámetro sin tipo que se utilizan en lugares concretos de una sentencia SQL. La tabla 1 muestra una lista de restricciones relacionadas con el uso de los marcadores de parámetros.

Tabla 8. Restricciones relacionadas con el uso de marcadores de parámetros

Ubicación del marcador de parámetro sin tipo	**Tipo de datos**
Expresión: solo en una lista de selección	Error
Expresión: ambos operandos de un operador aritmético	Error
Predicado: operando del lado izquierdo de un predicado IN	Error
Predicado: ambos operandos de un operador racional	Error
Función: operando de una función de agregación	Error

Ejemplos

DB2® proporciona un amplio conjunto de interfaces estándar que incluye CLI/ODBC, JDBC y ADO.NET para acceder a los datos de forma eficiente. Los siguientes fragmentos de código muestran sentencias preparadas con marcadores de parámetro para cada API de acceso de datos.

Considere el siguiente esquema de tabla para la tabla t1, donde la columna c1 es la clave primaria de la tabla t1.

Tabla 9. Esquema de tabla de ejemplo

Nombre de columna	**Tipo de datos de DB2**	**Puede contener nulos**
c1	INTEGER	false
c2	SMALLINT	true
c3	CHAR(20)	true
c4	VARCHAR(20)	true
c5	DECIMAL(8,2)	true
c6	DATE	true
c7	TIME	true
c8	TIMESTAMP	true
c9	BLOB(30)	true

En el siguiente ejemplo se muestra la forma en que insertar una fila en la tabla 1 utilizando una sentencia preparada.

Ejemplo de CLI

```
void parameterExample1(void)
{
   SQLHENV henv;
   SQLHDBC hdbc;
   SQLHSTMT hstmt;
   SQLRETURN rc;
   TCHAR server[] = _T("C:\\mysample\\");
   TCHAR uid[] = _T("db2e");
   TCHAR pwd[] = _T("db2e");
   long p1 = 10;
   short p2 = 100;
   TCHAR p3[100];
```

```
TCHAR p4[100];
TCHAR p5[100];
TCHAR p6[100];
TCHAR p7[100];
TCHAR p8[100];
char  p9[100];
long len = 0;

_tcscpy(p3, _T("data1"));
_tcscpy(p4, _T("data2"));
_tcscpy(p5, _T("10.12"));
_tcscpy(p6, _T("2003-06-30"));
_tcscpy(p7, _T("12:12:12"));
_tcscpy(p8, _T("2003-06-30-17.54.27.710000"));

memset(p9, 0, sizeof(p9));
p9[0] = 'X';
p9[1] = 'Y';
p9[2] = 'Z';

rc = SQLAllocEnv(&henv);
// verificar código de retorno ...

rc = SQLAllocConnect(henv, &hdbc);
// verificar código de retorno ...

rc = SQLConnect(hdbc, (SQLTCHAR*)server, SQL_NTS,
 (SQLTCHAR*)uid, SQL_NTS, (SQLTCHAR*)pwd, SQL_NTS);
// verificar código de retorno ...

rc = SQLAllocStmt(hdbc, &hstmt);
// verificar código de retorno ...

// preparar la sentencia
rc = SQLPrepare(hstmt, _T("INSERT INTO t1 VALUES (?,?,?,?,?,?,?,?,?)"), SQL_NTS);
// verificar código de retorno ...

// vincular los parámetros de entrada
rc = SQLBindParameter(hstmt, (unsigned short)1, SQL_PARAM_INPUT,
 SQL_C_LONG, SQL_INTEGER, 4, 0, &p1, sizeof(p1), &len);
// verificar código de retorno ...

rc = SQLBindParameter(hstmt, (unsigned short)2, SQL_PARAM_INPUT, SQL_C_LONG,
  SQL_SMALLINT, 2, 0, &p2, sizeof(p2), &len);
// verificar código de retorno ...

len = SQL_NTS;
rc = SQLBindParameter(hstmt, (unsigned short)3, SQL_PARAM_INPUT, SQL_C_TCHAR,
  SQL_CHAR, 0, 0, &p3[0], 100, &len);
// verificar código de retorno ...

rc = SQLBindParameter(hstmt, (unsigned short)4, SQL_PARAM_INPUT, SQL_C_TCHAR,
  SQL_VARCHAR, 0, 0, &p4[0], 100, &len);
// verificar código de retorno ...

rc = SQLBindParameter(hstmt, (unsigned short)5, SQL_PARAM_INPUT, SQL_C_TCHAR,
  SQL_DECIMAL, 8, 2, &p5[0], 100, &len);
// verificar código de retorno ...

rc = SQLBindParameter(hstmt, (unsigned short)6, SQL_PARAM_INPUT, SQL_C_TCHAR,
  SQL_TYPE_DATE, 0, 0, &p6[0], 100, &len);
// verificar código de retorno ...

rc = SQLBindParameter(hstmt, (unsigned short)7, SQL_PARAM_INPUT, SQL_C_TCHAR,
  SQL_TYPE_TIME, 0, 0, &p7[0], 100, &len);
// verificar código de retorno ...
```

```
   rc = SQLBindParameter(hstmt, (unsigned short)8, SQL_PARAM_INPUT, SQL_C_TCHAR,
     SQL_TYPE_TIMESTAMP, 0, 0, &p8[0], 100, &len);
   // verificar código de retorno ...

   len = 3;
   rc = SQLBindParameter(hstmt, (unsigned short)9, SQL_PARAM_INPUT, SQL_C_BINARY,
     SQL_BINARY, 0, 0, &p9[0], 100, &len);
   // verificar código de retorno ...

   // ejecutar la sentencia preparada
   rc = SQLExecute(hstmt);
   // verificar código de retorno ...

   rc = SQLFreeStmt(hstmt, SQL_DROP);
   // verificar código de retorno ...

   rc = SQLDisconnect(hdbc);
   // verificar código de retorno ...

   rc = SQLFreeConnect(hdbc);
   // verificar código de retorno ...

   rc = SQLFreeEnv(henv);
   // verificar código de retorno ...
```

Ejemplo de C

```
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;
```

Ejemplo de JDBC

```
public static void parameterExample1() {

   String driver = "com.ibm.db2e.jdbc.DB2eDriver";
   String url    = "jdbc:db2e:mysample";
   Connection conn = null;
   PreparedStatement pstmt = null;

   try
   {
      Class.forName(driver);

      conn = DriverManager.getConnection(url);

      // preparar la sentencia
      pstmt = conn.prepareStatement("INSERT INTO t1 VALUES
                                                  (?, ?, ?, ?, ?, ?, ?, ?, ?)");

      // vincular los parámetros de entrada
      pstmt.setInt(1, 1);
      pstmt.setShort(2, (short)2);
      pstmt.setString(3, "data1");
      pstmt.setString(4, "data2");
      pstmt.setBigDecimal(5, new java.math.BigDecimal("12.34"));
      pstmt.setDate(6, new java.sql.Date(System.currentTimeMillis() ) );
      pstmt.setTime(7, new java.sql.Time(System.currentTimeMillis() ) );
      pstmt.setTimestamp (8, new java.sql.Timestamp(System.currentTimeMillis() ) );
```

```
      pstmt.setBytes(9, new byte[] { (byte)'X', (byte)'Y', (byte)'Z' } );

      // ejecutar la sentencia preparada
      pstmt.execute();

      pstmt.close();

      conn.close();
   }
   catch (SQLException sqlEx)
   {
      while(sqlEx != null)
      {
          System.out.println("SQLERROR: \n" + sqlEx.getErrorCode() +
             ", SQLState: " + sqlEx.getSQLState() +
             ", Mensaje: " + sqlEx.getMessage() +
             ", Proveedor: " + sqlEx.getErrorCode() );
         sqlEx = sqlEx.getNextException();
      }
   }
   catch (Exception ex)
   {
      ex.printStackTrace();
   }
}
```

Ejemplo ADO.NET [C#]

```
public static void ParameterExample1()
{
   DB2eConnection conn = null;
   DB2eCommand cmd  = null;
   String connString   = @"database=.\; uid=db2e; pwd=db2e";
   int i = 1;

   try
   {
      conn = new DB2eConnection(connString);

      conn.Open();

      cmd = new DB2eCommand("INSERT INTO t1 VALUES
                                    (?, ?, ?, ?, ?, ?, ?, ?, ?)", conn);

      // preparar el mandato
      cmd.Prepare();

      // vincular los parámetros de entrada
      DB2eParameter p1 = new DB2eParameter("@p1", DB2eType.Integer);
      p1.Value = ++i;
      cmd.Parameters.Add(p1);

      DB2eParameter p2 = new DB2eParameter("@p2", DB2eType.SmallInt);
      p2.Value = 100;
      cmd.Parameters.Add(p2);

      DB2eParameter p3 = new DB2eParameter("@p3", DB2eType.Char);
      p3.Value = "data1";
      cmd.Parameters.Add(p3);

      DB2eParameter p4 = new DB2eParameter("@p4", DB2eType.VarChar);
      p4.Value = "data2";
      cmd.Parameters.Add(p4);

      DB2eParameter p5 = new DB2eParameter("@p5", DB2eType.Decimal);
      p5.Value = 20.25;
      cmd.Parameters.Add(p5);
```

```
      DB2eParameter p6 = new DB2eParameter("@p6", DB2eType.Date);
      p6.Value = DateTime.Now;
      cmd.Parameters.Add(p6);

      DB2eParameter p7 = new DB2eParameter("@p7", DB2eType.Time);
      p7.Value = new TimeSpan(23, 23, 23);
      cmd.Parameters.Add(p7);

      DB2eParameter p8 = new DB2eParameter("@p8", DB2eType.Timestamp);
      p8.Value = DateTime.Now;
      cmd.Parameters.Add(p8);

      byte []barr = new byte[3];
      barr[0] = (byte)'X';
      barr[1] = (byte)'Y';
      barr[2] = (byte)'Z';

      DB2eParameter p9 = new DB2eParameter("@p9", DB2eType.Blob);
      p9.Value = barr;
      cmd.Parameters.Add(p9);

      // ejecutar el mandato preparado
      cmd.ExecuteNonQuery();
   }
   catch (DB2eException e1)
   {
      for (int i=0; i < e1.Errors.Count; i++)
      {
         Console.WriteLine("Error núm:" + i + "\n" +
            "Mensaje: " + e1.Errors[i].Message + "\n" +
            "Nativo: " + e1.Errors[i].NativeError.ToString() + "\n" +
            "SQL: " + e1.Errors[i].SQLState + "\n");
      }
   }
   catch (Exception e2)
   {
       Console.WriteLine(e2.Message);
   }
   finally
   {
      if (conn != null && conn.State != ConnectionState.Closed)
      {
            conn.Close();
            conn = null;
      }
   }
}
```

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

El soporte de variables locales en los procedimientos de SQL le permite asignar y recuperar valores de SQL como ayuda a la lógica de procedimientos de SQL.

Las variables en los procedimientos de SQL se definen utilizando la sentencia DECLARE.

Pueden asignarse valores a variables utilizando la sentencia SET o la sentencia SELECT INTO o como un valor por omisión cuando se declara la variable. Pueden asignarse literales, expresiones, el resultado de una consulta y valores de registro especiales a las variables.

Pueden asignarse valores de variables a parámetros de procedimientos de SQL, a otras variables del procedimiento de SQL y se les puede hacer referencia como parámetros dentro de sentencias de SQL que se ejecuten dentro de la rutina.

El ejemplo siguiente demuestra diversos métodos para asignar y recuperar valores de variables.

```
CREATE PROCEDURE proc_vars()
SPECIFIC proc_vars
LANGUAGE SQL
BEGIN

  DECLARE v_rcount INTEGER;

  DECLARE v_max DECIMAL (9,2);

  DECLARE v_adate, v_another  DATE;

  DECLARE v_total INTEGER DEFAULT 0;  -- (1)

  SET v_total = v_total + 1                   -- (2)

  SELECT MAX(salary)                          -- (3)
    INTO v_max FROM employee;

  VALUES CURRENT_DATE INTO v_date;            -- (4)

  SELECT CURRENT DATE, CURRENT DATE           -- (5)
       INTO v_adate, v_another
  FROM SYSIBM.SYSDUMMY1;

  DELETE FROM T;
  GET DIAGNOSTICS v_rcount = ROW_COUNT;       -- (6)

END
```

Al declarar una variable, puede especificar un valor por omisión utilizando la cláusula DEFAULT como en la línea (1). La línea (2) muestra que puede utilizarse una sentencia SET para asignar un solo valor de variable. También pueden establecerse variables ejecutando una sentencia SELECT o FETCH en combinación con la cláusula INTO, como se muestra en la línea (3). Las líneas (4) y (5) muestran cómo la sentencia VALUES INTO puede utilizarse para evaluar una función o registro especial y asignar el valor a una variable o a múltiples variables.

También puede asignar el resultado de una sentencia GET DIAGNOSTICS a una variable. Puede utilizarse GET DIAGNOSTICS para manejar el número (#) de filas afectadas (actualizado para una sentencia UPDATE, DELETE para una sentencia DELETE) o para obtener el estado de retorno de una sentencia de SQL recién ejecutada. La línea (6) muestra cómo el número de filas modificadas por la sentencia DELETE recién ejecutada puede asignarse a una variable.

Soporte de XML y XQuery en procedimientos de SQL

Los procedimientos de SQL soportan parámetros y variables de tipo de datos XML. También puede utilizarse en las sentencias de SQL, del mismo modo que las variables de cualquier otro tipo de datos. Además, las variables del tipo de datos XML pueden pasarse como parámetros para expresiones XQuery en expresiones XMLEXISTS, XMLQUERY y XMLTABLE.

En el siguiente ejemplo se muestra la declaración, el uso y la asignación de los parámetros y variables XML en un procedimiento de SQL:

```
CREATE TABLE T1(C1 XML) %

CREATE PROCEDURE proc1(IN parm1 XML, IN parm2 VARCHAR(32000))
LANGUAGE SQL
BEGIN
   DECLARE var1 XML;

   /* comprobar si el valor del parámetro XML parm1 contiene un elemento con
      un valor menor que 200 */
   IF(XMLEXISTS('$x/ITEM[value < 200]' passing by ref parm1 as "x"))THEN

     /* si lo contiene, insertar el valor de parm1 en la tabla T1 */
     INSERT INTO T1 VALUES(parm1);

   END IF;

   /* analizar valor del parámetro parm2 y asignarlo a una variable */
   SET var1 = XMLPARSE(document parm2 preserve whitespace);

   /* insertar variable var1 en tabla T1
   INSERT INTO T1 VALUES(var1);

END %
```

En el ejemplo anterior, hay una tabla `T1` con una columna XML. El procedimiento de SQL acepta dos parámetros de tipo de datos XML que se llaman `parm1` y `parm2`. Dentro del procedimiento de SQL, se declara una variable XML que se llama `var1`.

La lógica del procedimiento de SQL comprueba si el valor del parámetro XML `parm1` contiene un elemento con un valor menor que 200. Si lo contiene, el valor XML se inserta directamente en la columna `C1` de la tabla `T1`.

Luego se analiza el valor del parámetro `parm2` mediante la función XMLPARSE y se asigna a la variable XML `var1`. Después, el valor de la variable XML también se inserta en la columna `C1` de la tabla `T1`.

La capacidad para implementar lógica de flujo de control alrededor de operaciones XQuery hace que resulte fácil desarrollar algoritmos complejos que consulten datos XML almacenados en una base de datos y accedan a ellos.

Variables SQLCODE y SQLSTATE en procedimientos de SQL

Para llevar a cabo el manejo de errores o ayudarle a depurar los procedimientos de SQL, tal vez le resulte útil comprobar el valor de los valores de SQLCODE o SQLSTATE, devolver esos valores como parámetros de salida o como parte de una serie de mensajes de diagnóstico, o insertarlos en una tabla para proporcionar un soporte de rastreo básico.

Para utilizar los valores de SQLCODE y SQLSTATE en los procedimientos de SQL, debe declarar las siguientes variables de SQL en el cuerpo del procedimiento de SQL:

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

DB2 establece implícitamente estas variables siempre que se ejecuta una sentencia. Si una sentencia produce una condición para la que existe un manejador, los valores de las variables SQLSTATE y SQLCODE estarán disponibles al iniciarse la ejecución del manejador. No obstante, las variables se restablecerán tan pronto como se ejecute la primera sentencia del manejador. Por lo tanto, es una práctica habitual copiar los valores de SQLSTATE y SQLCODE en variables locales en la

primera sentencia del manejador. En el ejemplo siguiente, se utiliza un manejador CONTINUE para cualquier condición a fin de copiar la variable SQLCODE en otra variable denominada `retcode`. La variable `retcode` se puede utilizar entonces en las sentencias ejecutables para controlar la lógica de procedimientos o devolver el valor como parámetro de salida.

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

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

  sentencias-ejecutables
END
```

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

Sentencias compuestas y ámbito de las variables en los procedimientos de SQL

Los procedimientos de SQL pueden contener una o varias sentencias compuestas. Pueden presentarse en series o pueden anidarse dentro de otra sentencia compuesta. Cada sentencia compuesta presenta un nuevo ámbito en el que tal vez haya variables disponibles para su utilización o tal vez no.

La utilización de etiquetas para identificar una sentencia compuesta es importante ya que la etiqueta puede utilizarse para calificar e identificar de manera exclusiva variables declaradas dentro de la sentencia compuesta. Esto es especialmente importante cuando se haga referencia a variables que se encuentren en sentencias compuestas distintas o en sentencias compuestas anidadas.

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

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

      SET a = a + lab1.a;

      UPDATE T1
       SET T1.b = 5
        WHERE T1.b = a;  <-- La variable a hace referencia a lab2.a
                             a menos que esté calificada de otra forma

    END lab2;
 END lab1
```

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

Efecto de los compromisos y retrotracciones en los valores de variables y parámetros XML de los procedimientos de SQL

Los compromisos y retrotracciones de los procedimientos de SQL afectan a los valores de los parámetros y variables de tipo de datos XML. Durante la ejecución de los procedimientos de SQL, al producirse una operación de compromiso o retrotracción, los valores asignados a las variables XML y a los parámetros XML dejarán de estar disponibles.

Los intentos de hacer referencia a una variable SQL o a un parámetro SQL de tipo de datos XML después de una operación de compromiso o retrotracción provocarán un error (SQL1354N, 560CE).

Para hacer referencia de modo satisfactorio a parámetros y variables XML después de que se produzca una operación de compromiso o retrotracción, en primer lugar, debe asignárseles nuevos valores.

Tenga en cuenta la disponibilidad de los valores de las variables y parámetros XML al añadir sentencias ROLLBACK y COMMIT a los procedimientos de SQL.

Cursores en los procedimientos de SQL

En los procedimientos de SQL, un cursor permite definir un conjunto de resultados (un conjunto de filas de datos) y ejecutar lógica compleja fila a fila. Utilizando la misma mecánica, un procedimiento de SQL también puede definir un conjunto de resultados y devolverlo directamente a quien llamó al procedimiento de SQL o a una aplicación cliente.

Un cursor puede considerarse como un puntero a una fila de un conjunto de filas. El cursor sólo puede hacer referencia a una fila a la vez, pero puede moverse a otras filas del conjunto de resultados según sea necesario.

Para utilizar cursores en procedimientos de SQL, debe hacer lo siguiente:

1. Declare un cursor que defina un conjunto de resultados.
2. Abra el cursor para establecer el conjunto de resultados.
3. Extraiga los datos del cursor en variables locales según sea necesario, una fila cada vez.
4. Cierre el cursor cuando hay terminado

Para trabajar con cursores, debe utilizar las siguientes sentencias de SQL:

- DECLARE CURSOR
- OPEN
- FETCH
- CLOSE

El ejemplo siguiente muestra la utilización básica de un cursor de sólo lectura dentro de un procedimiento de SQL:

```
CREATE PROCEDURE sum_salaries(OUT sum INTEGER)
LANGUAGE SQL
BEGIN
```

```
    DECLARE p_sum INTEGER;
    DECLARE p_sal INTEGER;
    DECLARE c CURSOR FOR SELECT SALARY FROM EMPLOYEE;
    DECLARE SQLSTATE CHAR(5) DEFAULT '00000';

     SET p_sum = 0;

     OPEN c;

     FETCH FROM c INTO p_sal;

     WHILE(SQLSTATE = '00000') DO
        SET p_sum = p_sum + p_sal;
        FETCH FROM c INTO p_sal;
     END WHILE;

     CLOSE c;

     SET sum = p_sum;

  END%
```

Este es un ejemplo más complejo de utilización de un cursor dentro de un procedimiento de SQL. Este ejemplo muestra el uso combinado de un cursor y sentencias PL de SQL.

Cursores para expresiones XQuery en procedimientos de SQL

Los procedimientos de SQL soportan la definición de cursores en expresiones Xquery. Un cursor en una expresión XQuery le permite repetir los elementos de la secuencia XQuery que devuelve la expresión.

Al contrario que los cursores definidos en las sentencias de SQL, que pueden definirse estática o dinámicamente, los cursores de las expresiones XQuery sólo pueden definirse dinámicamente. Para declarar un cursor dinámicamente, es necesario que declare una variable del tipo CHAR o VARCHAR para que contenga la expresión XQuery que definirá el conjunto de resultados del cursor. La expresión XQuery debe estar preparada antes de que pueda abrirse el cursor y se resuelva el conjunto de resultados.

A continuación se muestra un ejemplo de procedimiento de SQL que declara dinámicamente un cursor para una expresión XQuery, abre el cursor y capta datos XML:

```
CREATE PROCEDURE xmlProc(IN inCust XML, OUT resXML XML)
SPECIFIC xmlProc
LANGUAGE SQL
BEGIN
    DECLARE SQLSTATE CHAR(5);
    DECLARE stmt_text VARCHAR (1024);
    DECLARE customer XML;
    DECLARE cityXml XML;
    DECLARE city VARCHAR (100);
    DECLARE stmt STATEMENT;
    DECLARE cur1 CURSOR FOR stmt;

   -- Get the city of the input customer
   SET cityXml = XMLQUERY('$cust/customerinfo//city' passing inCust as "cust");
   SET city = XMLCAST(cityXml as VARCHAR(100));

  -- Iterate over all the customers from the city using an XQUERY cursor
  -- and collect the customer name values into the output XML value

   SET stmt_text = 'XQUERY for $cust
```

```
                    in db2-fn:xmlcolumn("CUSTOMER.INFO")
                       /*:customerinfo/*:addr[*:city= "' || city ||'"]
                    return <Customer>{$cust/../@Cid}{$cust/../*:name}</Customer>';

  -- Use the name of the city for the input customer data as a prefix
  SET resXML = cityXml;

  PREPARE stmt FROM stmt_text;
  OPEN cur1;

  FETCH cur1 INTO customer;
  WHILE(SQLSTATE = '00000') DO
    SET resXML = XMLCONCAT(resXML, customer);
    FETCH cur1 INTO customer;
  END WHILE;

  set resXML = XMLQUERY('<result> {$res} </result>'
                        passing resXML as "res");

END
```

Este procedimiento de SQL recopila los ID y nombres de clientes definidos en el nombre de tabla CUSTOMER que están ubicados en la misma ciudad que el cliente para el que se han proporcionado datos XML como parámetro de entrada.

El procedimiento de SQL anterior puede llamarse ejecutando la sentencia CALL del siguiente modo:

```
  CALL xmlProc(xmlparse(document '<customerinfo Cid="5002">
                                    <name>Jim Noodle</name>
                                    <addr country="Canada">
                                       <street>25 EastCreek</street>
                                       <city>Markham</city>
                                       <prov-state>Ontario</prov-state>
                                       <pcode-zip>N9C-3T6</pcode-zip>
                                    </addr>
                                    <phone type="work">905-566-7258</phone>
                                  </customerinfo>' PRESERVE WHITESPACE),?)
```

Si se crea este procedimiento de SQL y se ejecuta en la base de datos SAMPLE, devuelve datos XML para dos clientes.

Puesto que los valores XML no dan soporte a los marcadores de parámetros, un método alternativo a esta limitación es construir una sentencia SQL dinámica compuesta de fragmentos concatenados de sentencias que incluyan el valor de una o más variables locales.

Por ejemplo:

```
DECLARE person_name VARCHAR(128);

SET person_name = "Joe";
SET stmt_text = 'XQUERY for $fname in db2-fn:sqlquery
   ("SELECT doc
      FROM T1
      WHERE DOCID=1")//fullname where $fname/first = ''' person_name || ''';
```

Este ejemplo devuelve un conjunto de resultados en una asignación de variables para una sentencia XQuery que contiene una selección completa de SQL. El conjunto de resultados contiene los nombres completos de las personas cuyo nombre es Joe. Funcionalmente, la parte de SQL selecciona los documentos XML

de la columna doc en la tabla T1 que tiene un ID de 1. La parte de XQuery selecciona a continuación los valores de fullname en los documentos XML en los que el valor first es Joe.

Elementos de lógica SQL PL en el cuerpo del procedimiento de SQL

La ejecución secuencial es la ruta más básica que puede tomar la ejecución del programa. Con este método, el programa inicia la ejecución en la primera línea del código, seguido de la siguiente, y continúa hasta que se haya ejecutado la sentencia final del código. Este enfoque funciona bien para tareas simples, pero no resulta muy útil ya que solamente puede manejar una situación. Los programas necesitan a menudo poder decidir qué hacer como respuesta a los cambios en las circunstancias. Controlando la ruta de ejecución de un código, una parte de código específica puede utilizarse entonces para manejar de manera inteligente más de una situación.

SQL PL proporciona soporte para variables y el flujo de sentencias de control que pueden utilizarse para controlar la secuencia de ejecución de sentencias. Las sentencias como IF y CASE se utilizan para ejecutar de forma condicional bloques de sentencias de SQL PL, mientras otras sentencias, tales como WHILE y REPEAT, se utilizan normalmente para ejecutar un conjunto de sentencias de manera repetitiva hasta que se completa una tarea.

Aunque existen muchos tipos de sentencias de SQL PL, pueden clasificarse en unas pocas categorías:

- Sentencias relacionadas con variables
- Sentencias condicionales
- Sentencias en bucle
- Sentencias de transferencia de control

Sentencias relacionadas con variables en procedimientos de SQL

Las sentencias de SQL relacionadas con variables se utilizan para declarar variables y para asignar valores a las variables. Existen algunos tipos de sentencias relacionadas con variables:

- Sentencia DECLARE <variable> en procedimientos de SQL
- Sentencia DECLARE <condición> en procedimientos de SQL
- Sentencia DECLARE <descriptor de contexto de condición> en procedimientos de SQL
- DECLARE CURSOR en procedimientos de SQL
- SET (sentencia-asignación) en procedimientos de SQL

Estas sentencias proporcionan el soporte necesario para utilizar los demás tipos de sentencias PL de SQL y las sentencias de SQL que utilizarán valores de variable.

Sentencias condicionales en procedimientos de SQL

Las sentencias condicionales se utilizan para definir la lógica que debe ejecutarse en función del estado de alguna condición que debe cumplirse. Existen dos tipos de sentencias condicionales soportadas en procedimientos de SQL:

- CASE
- IF

Estas sentencias son similares, pero la sentencia CASE amplía la sentencia IF.

Sentencia CASE en procedimientos de SQL

Las sentencias CASE pueden utilizarse para entrar condicionalmente en una lógica determinada en función del estado de una condición que se cumple. Existen dos tipos de sentencias CASE:

- Sentencia case simple: utilizada para entrar en una lógica determinada en función de un valor literal.
- Sentencia case buscada: utilizada para entrar en una lógica determinada en función del valor de una expresión.

La cláusula WHEN de la sentencia CASE define el valor que, si se cumple, determina el flujo de control.

A continuación figura un ejemplo de un procedimiento de SQL con una sentencia CASE con una cláusula WHEN de sentencia CASE simple:

```
CREATE PROCEDURE UPDATE_DEPT (IN p_workdept)
LANGUAGE SQL
BEGIN

  DECLARE v_workdept CHAR(3);
  SET v_workdept = p_workdept;

  CASE  v_workdept
    WHEN 'A00' THEN
      UPDATE department SET deptname = 'D1';
    WHEN 'B01' THEN
      UPDATE department SET deptname = 'D2';
    ELSE
      UPDATE department SET deptname = 'D3';
    END CASE

END
```

A continuación figura un ejemplo de una sentencia CASE con una cláusula WHEN de sentencia CASE buscada:

```
CREATE PROCEDURE UPDATE_DEPT (IN p_workdept)
LANGUAGE SQL
BEGIN

  DECLARE v_workdept CHAR(3);
  SET v_workdept = p_workdept;

  CASE
    WHEN v_workdept = 'A00' THEN
      UPDATE department SET deptname = 'D1';
    WHEN v_workdept = 'B01' THEN
      UPDATE department SET deptname = 'D2';
    ELSE
      UPDATE department SET deptname = 'D3';
  END CASE

END
```

Los ejemplos suministrados anteriormente son lógicamente equivalentes, pero es importante tener en cuenta que las sentencias CASE con una cláusula WHEN de sentencia CASE buscada pueden ser muy potentes. En este caso puede utilizarse cualquier expresión SQL soportada. Estas expresiones pueden contener referencias a variables, parámetros, registros especiales, etc.

Sentencia IF en procedimientos de SQL

Las sentencias IF pueden utilizarse para entrar condicionalmente en una lógica determinada en función del estado de una condición que se cumple. La sentencia IF es lógicamente equivalente a una sentencia CASE con una cláusula searched-case-statement-when.

La sentencia IF da soporte a la utilización de cláusulas ELSE IF opcionales y a una cláusula ELSE predeterminada. Una cláusula END IF es necesaria para indicar el final de la sentencia.

A continuación figura un ejemplo de procedimiento que contiene una sentencia IF:

```
CREATE PROCEDURE UPDATE_SAL (IN empNum CHAR(6),
                             INOUT rating SMALLINT)
LANGUAGE SQL
BEGIN
  IF rating = 1 THEN
    UPDATE employee
    SET salary = salary * 1.10, bonus = 1000
      WHERE empno = empNum;
  ELSEIF rating = 2 THEN
    UPDATE employee
    SET salary = salary * 1.05, bonus = 500
      WHERE empno = empNum;
  ELSE
    UPDATE employee
    SET salary = salary * 1.03, bonus = 0
      WHERE empno = empNum;
  END IF;
END
```

Sentencias en bucle en procedimientos de SQL

Las sentencias en bucle proporcionan soporte para la ejecución repetida de una determinada lógica hasta que se cumple una condición. SQL PL da soporte a las siguientes sentencias en bucle:

- FOR
- LOOP
- REPEAT
- WHILE

La sentencia FOR es diferente de las demás, ya que se utiliza para la iteración en filas de un conjunto de resultados definido, mientras que las demás se utilizan para la iteración en una serie de sentencias de SQL hasta que se cumple una condición para cada una de ellas.

Pueden definirse etiquetas para todas las sentencias de control de bucle para identificarlas.

Sentencia FOR en procedimientos de SQL

La sentencia FOR es un tipo especial de sentencia de bucle, porque se utiliza para iterar filas de un conjunto de resultados de sólo lectura definido. Cuando se ejecuta una sentencia FOR, se declara implícitamente un cursor a fin de que, para cada iteración del bucle LOOP, la fila siguiente sea el conjunto de resultados si se extrae. El bucle continúa hasta que no quedan más filas en el conjunto de resultados.

La sentencia FOR simplifica la implementación de un cursor y facilita la recuperación de un conjunto de valores de columna para un conjunto de filas sobre el que pueden realizarse operaciones lógicas.

A continuación figura un ejemplo de un procedimiento de SQL que contiene sólo una sentencia FOR simple:

```
CREATE PROCEDURE P()
LANGUAGE SQL
BEGIN ATOMIC
  DECLARE fullname CHAR(40);

  FOR v AS cur1 CURSOR FOR
             SELECT firstnme, midinit, lastname FROM employee
  DO
    SET fullname = v.lastname || ',' || v.firstnme
                   ||' ' || v.midinit;
    INSERT INTO tnames VALUES (fullname);
  END FOR;
END
```

Nota: La mejor manera de implementar una lógica como la que se muestra en el ejemplo anterior es mediante la función CONCAT. El ejemplo sencillo sirve para mostrar la sintaxis.

El nombre de bucle for especifica una etiqueta para la sentencia compuesta implícita generada para implementar la sentencia FOR. Sigue las normas de las etiquetas de una sentencia compuesta. El nombre de bucle for puede utilizarse para calificar los nombres de columna del conjunto de resultados devuelto por la sentencia select.

El nombre de cursor denomina simplemente el cursor utilizado para seleccionar las filas del conjunto de resultados. i no se especifica, el gestor de bases de datos de DB2 generará automática e internamente un nombre de cursor exclusivo.

Los nombres de columna de la sentencia select deben ser exclusivos y es necesaria una cláusula FROM que especifique una tabla (o varias tablas si se realiza algún tipo de JOIN o UNION). Las tablas y columnas referenciadas deben existir antes de ejecutar el bucle. Puede referenciarse tablas temporales globales.

Las actualizaciones y supresiones de posición y las actualizaciones y supresiones de búsqueda están soportadas en el bucle FOR. Para asegurar que los resultados sean correctos, la especificación de cursor del bucle FOR debe incluir una cláusula FOR UPDATE.

El cursor que se crea para dar soporte a la sentencia FOR no puede referenciarse fuera del bucle FOR.

Sentencia LOOP en procedimientos de SQL

La sentencia LOOP es un tipo especial de sentencia de bucle, porque no tiene cláusula de terminación de condición. Define una serie de sentencias que se ejecutan repetidamente hasta que otra parte de la lógica, generalmente una sentencia de transferencia de control, fuerza al flujo de control a saltar a algún punto externo al bucle.

La sentencia LOOP se utiliza generalmente junto con una de las siguientes sentencias: LEAVE, GOTO, ITERATE o RETURN. Estas sentencias pueden forzar el control a inmediatamente después del bucle, a una ubicación especificada del procedimiento de SQL, al principio del bucle para empezar otra iteración del bucle

o para salir del procedimiento de SQL. Para indicar dónde debe pasarse el flujo al utilizar estas sentencias, se utilizan etiquetas.

La sentencia LOOP resulta de utilidad si tiene una lógica complicada en un bucle del que puede ser necesario salir de más de una forma, pero debe utilizarse con cuidado para evitar instancias de bucles infinitos.

Si la sentencia LOOP se utiliza autónomamente sin una sentencia de transferencia de control, la serie de sentencias incluidas en el bucle se ejecutará indefinidamente o hasta que se produzca una condición de base de datos que lance un manejador de condiciones que fuerce un cambio en el flujo de control o se produzca una condición no manejada que fuerce el retorno del procedimiento de SQL.

A continuación figura un ejemplo de un procedimiento de SQL que contiene una sentencia LOOP. También utiliza las sentencias ITERATE y LEAVE.

```
CREATE  PROCEDURE ITERATOR()
LANGUAGE SQL
BEGIN
  DECLARE v_deptno CHAR(3); DECLARE v_deptname VARCHAR(29);
  DECLARE at_end INTEGER DEFAULT 0;
  DECLARE not_found CONDITION FOR SQLSTATE '02000';

  DECLARE c1 CURSOR FOR SELECT deptno, deptname
                        FROM department ORDER BY deptno;
  DECLARE CONTINUE HANDLER FOR not_found SET at_end = 1;
  OPEN c1;

  ins_loop: LOOP

    FETCH c1 INTO v_deptno, v_deptname;

    IF at_end = 1 THEN
      LEAVE ins_loop;
    ELSEIF v_dept = 'D11' THEN
      ITERATE ins_loop;
    END IF;

    INSERT  INTO department (deptno, deptname)
      VALUES ('NEW', v_deptname);

  END LOOP;

  CLOSE c1;
END
```

Sentencia WHILE en procedimientos de SQL

La sentencia WHILE define un conjunto de sentencias que deben ejecutarse hasta que deja de cumplirse una condición que se evalúa al principio del bucle WHILE. La condición de bucle while (una expresión) se evalúa antes de cada iteración del bucle.

A continuación figura un ejemplo de un procedimiento de SQL con un bucle WHILE simple:

```
CREATE PROCEDURE sum_mn (IN p_start INT
                        ,IN p_end INT
                        ,OUT p_sum INT)
SPECIFIC sum_mn
LANGUAGE SQL
smn: BEGIN

DECLARE v_temp INTEGER DEFAULT 0;
DECLARE v_current INTEGER;
```

```
SET v_current = p_start;

WHILE (v_current <= p_end) DO
  SET v_temp = v_temp + v_current;
  SET v_current = v_current + 1;
END WHILE;
p_sum = v_current;
END smn;
```

Nota: La mejor manera de implementar la lógica como la que se muestra en el ejemplo anterior es mediante una fórmula matemática. El ejemplo sencillo sirve para mostrar la sintaxis.

Sentencia REPEAT en procedimientos de SQL

La sentencia REPEAT define un conjunto de sentencias que deben ejecutarse hasta que se cumple una condición que se evalúa al final del bucle REPEAT. La condición del bucle repeat se evalúa al final de cada iteración del bucle.

Con una sentencia WHILE, no se entra en el bucle si la condición de bucle while no se cumple en el primer paso. La sentencia REPEAT es una alternativa útil; sin embargo, debe tenerse en cuenta que la lógica del bucle while puede reescribirse como una sentencia REPEAT.

A continuación figura un procedimiento de SQL que incluye una sentencia REPEAT:

```
CREATE PROCEDURE sum_mn2 (IN p_start INT
                         ,IN p_end INT
                         ,OUT p_sum INT)
SPECIFIC sum_mn2
LANGUAGE SQL
smn2: BEGIN

  DECLARE v_temp INTEGER DEFAULT 0;
  DECLARE v_current INTEGER;

  SET v_current = p_start;

  REPEAT
    SET v_temp = v_temp + v_current;
    SET v_current = v_current + 1;
  UNTIL (v_current > p_end)
  END REPEAT;
END
```

Sentencias de transferencia de control en procedimientos de SQL

Las sentencias de transferencia de control se utilizan para redireccionar el flujo de control dentro de un procedimiento de SQL. Esta ramificación incondicional puede utilizarse para que el flujo de control salte de un punto a otro, que puede preceder o seguir a la sentencia de transferencia de control. Las sentencias de transferencia de flujo de control soportadas en procedimientos de SQL son las siguientes:

- GOTO
- ITERATE
- LEAVE
- RETURN

Las sentencias de transferencia de control pueden utilizarse en cualquier lugar de un procedimiento de SQL, pero ITERATE y LEAVE se utilizan generalmente junto con una sentencia LOOP u otras sentencias en bucle.

Sentencia GOTO en procedimientos de SQL

La sentencia GOTO es una sentencia de flujo de control directa y básica que provoca un cambio incondicional en el flujo de control. Se utiliza para la ramificación hacia una ubicación específica definida por usuario mediante etiquetas definidas en el procedimiento de SQL.

La utilización de la sentencia GOTO se considera generalmente una práctica de programación pobre y no se aconseja. La utilización extensiva de GOTO tiende a conducir a un código ilegible cuando los procedimientos crecen. Además, GOTO no es necesaria, porque existen mejores sentencias disponibles para controlar la vía de ejecución. No hay situaciones específicas que requieran la utilización de GOTO; con frecuencia se utiliza a efectos de conveniencia.

A continuación figura un ejemplo de un procedimiento de SQL que contiene una sentencia GOTO:

```
CREATE PROCEDURE adjust_salary ( IN p_empno CHAR(6),
                     IN p_rating INTEGER,
     OUT p_adjusted_salary DECIMAL (8,2) )
LANGUAGE SQL
BEGIN
  DECLARE new_salary DECIMAL (9,2);
  DECLARE service DATE;  -- start date

  SELECT salary, hiredate INTO v_new_salary, v_service
    FROM employee
       WHERE empno = p_empno;

  IF service > (CURRENT DATE - 1 year) THEN
    GOTO exit;
  END IF;

  IF p_rating = 1 THEN
    SET new_salary = new_salary + (new_salary * .10);
  END IF;

  UPDATE employee SET salary = new_salary WHERE empno = p_empno;

exit:
  SET p_adjusted_salary = v_new_salary;


END
```

Este ejemplo muestra una buena utilización de la sentencia GOTO: saltar casi al final de un procedimiento o bucle para que no se ejecuta una lógica determinada, pero asegurarse de que otra lógica determinada se siga ejecutando.

Debe tener en cuenta algunas consideraciones adicionales acerca del ámbito al utilizar la sentencia GOTO:

- Si la sentencia GOTO está definida en una sentencia FOR, la etiqueta debe definirse dentro de la misma sentencia FOR, a menos que esté en una sentencia FOR anidada o una sentencia compuesta anidada.
- Si la sentencia GOTO está definida en una sentencia compuesta, la etiqueta debe definirse dentro de la misma sentencia compuesta, a menos que esté en una sentencia FOR anidada o una sentencia compuesta anidada.

- Si la sentencia GOTO está definida en un manejador, la etiqueta debe definirse en el mismo manejador, siguiendo las demás normas de ámbito.
- Si la sentencia GOTO está definida fuera de un manejador, la etiqueta no debe definirse dentro de un manejador.
- Si la etiqueta no está definida dentro de un ámbito que la sentencia GOTO puede alcanzar, se devuelve un error (SQLSTATE 42736).

Sentencia ITERATE en procedimientos de SQL

La sentencia ITERATE se utiliza para que el flujo de control vuelva al principio de una sentencia LOOP etiquetada.

A continuación figura un ejemplo de un procedimiento de SQL que contiene una sentencia ITERATE:

```
CREATE  PROCEDURE ITERATOR()
LANGUAGE SQL
BEGIN
  DECLARE v_deptno CHAR(3); DECLARE v_deptname VARCHAR(29);
  DECLARE at_end INTEGER DEFAULT 0;
  DECLARE not_found CONDITION FOR SQLSTATE '02000';

  DECLARE c1 CURSOR FOR SELECT deptno, deptname
                        FROM department ORDER BY deptno;
  DECLARE CONTINUE HANDLER FOR not_found SET at_end = 1;
  OPEN c1;

  ins_loop: LOOP
   FETCH c1 INTO v_deptno, v_deptname;
      IF at_end = 1 THEN
        LEAVE ins_loop;
      ELSEIF v_dept = 'D11' THEN
        ITERATE ins_loop;
      END IF;


      INSERT  INTO department (deptno, deptname)
      VALUES ('NEW', v_deptname);

  END LOOP;

  CLOSE c1;

END
```

En el ejemplo, la sentencia ITERATE se utiliza para devolver el flujo de control a la sentencia LOOP definida con la etiqueta ins_loop cuando un valor de columna de una fila extraída coincide con un valor determinado. La posición de la sentencia ITERATE garantiza que no se inserten valores en la tabla department.

Sentencia LEAVE en procedimientos de SQL

La sentencia LEAVE se utiliza para transferir el flujo de control desde un bucle o sentencia compuesta.

A continuación figura un ejemplo de un procedimiento de SQL que contiene una sentencia LEAVE:

```
CREATE  PROCEDURE ITERATOR()
LANGUAGE SQL
BEGIN
  DECLARE v_deptno CHAR(3); DECLARE v_deptname VARCHAR(29);
  DECLARE at_end INTEGER DEFAULT 0;
  DECLARE not_found CONDITION FOR SQLSTATE '02000';
```

```
    DECLARE c1 CURSOR FOR SELECT deptno, deptname
                          FROM department ORDER BY deptno;
    DECLARE CONTINUE HANDLER FOR not_found SET at_end = 1;

    OPEN c1;

    ins_loop: LOOP

      FETCH c1 INTO v_deptno, v_deptname;

      IF at_end = 1 THEN
        LEAVE ins_loop;
      ELSEIF v_dept = 'D11' THEN
        ITERATE ins_loop;
      END IF;

      INSERT  INTO department (deptno, deptname)
      VALUES ('NEW', v_deptname);

    END LOOP;

    CLOSE c1;
END
```

En el ejemplo, la sentencia LEAVE se utiliza para salir de la sentencia LOOP definida con la etiqueta ins_loop. Está anidada dentro de una sentencia IF y, por tanto, se ejecuta condicionalmente cuando se cumple la condición IF, que se cumple cuando no se encuentran más filas en el cursor. La posición de la sentencia LEAVE garantiza que no se ejecutan más iteraciones del bucle una vez que se emite un error de tipo NOT FOUND.

Sentencia RETURN en procedimientos de SQL

La sentencia RETURN se utiliza para terminar incondicionalmente e inmediatamente un procedimiento de SQL devolviendo el flujo de control al llamador del procedimiento almacenado.

Es obligatorio que la ejecución de la sentencia RETURN devuelva un valor entero. Si el valor de retorno no se suministra, el valor predeterminado es 0. El valor se utiliza generalmente para indicar el éxito o el fracaso de la ejecución del procedimiento. El valor puede ser un literal, una variable o una expresión que se evalúe en un valor entero.

Puede utilizar una o varias sentencias RETURN en un procedimiento almacenado. La sentencia RETURN puede utilizarse en cualquier lugar después de los bloques de declaración dentro del cuerpo del procedimiento de SQL.

Para devolver varios valores de salida, pueden utilizarse parámetros. Los valores de parámetro deben establecerse antes de ejecutar la sentencia RETURN.

A continuación figura un ejemplo de un procedimiento de SQL que utiliza la sentencia RETURN:

```
  CREATE PROCEDURE return_test (IN p_empno CHAR(6),
                                IN p_emplastname VARCHAR(15) )
  LANGUAGE SQL
  SPECIFIC return_test
  BEGIN

    DECLARE v_lastname VARCHAR (15);

    SELECT lastname INTO v_lastname
      FROM employee
```

```
  WHERE empno = p_empno;

  IF v_lastname = p_emplastname THEN
    RETURN 1;
  ELSE
    RETURN -1;
  END IF;

END rt
```

En el ejemplo, si el parámetro *p_emplastname* coincide con el valor almacenado en table employee, el procedimiento devuelve 1. Si no coincide, devuelve -1.

Manejadores de condiciones en los procedimientos de SQL

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

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

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

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

La sintaxis de declaración del descriptor de contexto para los descriptores de contexto de condición se describe en la sentencia (procedimiento) de SQL compuesto.

Devolución de conjuntos de resultados desde procedimientos de SQL

En los procedimientos de SQL, los cursores pueden utilizarse para más que iterar por filas de un conjunto de resultados. También pueden utilizarse para devolver conjuntos de resultados al programa de llamada. Los procedimientos de SQL pueden recuperar conjuntos de resultados (en el caso de llamadas de procedimientos anidados o aplicaciones cliente programadas en C utilizando la interfaz de programación de aplicaciones CLI, lenguajes Java, CLI o .NET CLR.

Requisitos previos

- Autorización para crear un procedimiento de SQL

Para devolver un conjunto de resultados de un procedimiento de SQL, debe:

1. Especificar la cláusula DYNAMIC RESULT SETS en la sentencia CREATE PROCEDURE
2. DECLARE el cursor utilizando la cláusula WITH RETURN
3. Abrir el cursor en el procedimiento de SQL
4. Mantener el cursor abierto para la aplicación de cliente; no lo cierre

A continuación se proporciona un ejemplo de un procedimiento de SQL que solo devuelve un conjunto de resultados:

```
CREATE PROCEDURE  read_emp()
SPECIFIC read_emp
LANGUAGE SQL
DYNAMIC RESULT SETS 1

Re:  BEGIN

  DECLARE c_emp CURSOR WITH RETURN FOR
    SELECT salary, bonus, comm.
    FROM employee
    WHERE job != 'PRES';

  OPEN c_emp;

END Re
```

Si el cursor se cierra utilizando la sentencia CLOSE antes del retorno del procedimiento de SQL, el conjunto de resultados del cursor no se devolverá al llamador o a la aplicación de cliente.

Pueden devolverse múltiples conjuntos de resultados de un procedimiento de SQL utilizando múltiples cursores. Para devolver múltiples cursores, debe realizarse lo siguiente:

- Especificar la cláusula DYNAMIC RESULT SETS en la sentencia CREATE PROCEDURE. Especifique el número máximo posible de conjuntos de resultados que podrían devolverse. El número de conjuntos de resultados devueltos realmente no debe sobrepasar este número.
- Declarar cursores para cada uno de los conjuntos de resultados a devolver que especifiquen la cláusula WITH RETURN.
- Abrir los cursores a devolver.
- Mantener el cursor abierto para la aplicación de cliente; no lo cierre.

Es necesario un cursor por cada conjunto de resultados que deba devolverse.

Los conjuntos de resultados se devuelven al llamador en el orden en que se abren.

Una vez haya creado el procedimiento de SQL que devuelve un conjunto de resultados, puede interesarle llamarlo y recuperar el conjunto de resultados.

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

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

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

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

1. Debe declarar (DECLARE) localizadores de conjunto de resultados para cada conjunto de resultados que devuelva el procedimiento. Por ejemplo:

   ```
   DECLARE result1 RESULT_SET_LOCATOR VARYING;
   DECLARE result2 RESULT_SET_LOCATOR VARYING;
   DECLARE result3 RESULT_SET_LOCATOR VARYING;
   ```

2. Invoque el procedimiento. Por ejemplo:

   ```
   CALL targetProcedure();
   ```

3. Asocie (ASSOCIATE) las variables de localizador de conjunto de resultados (definidas anteriormente) con el procedimiento invocado. Por ejemplo:

   ```
   ASSOCIATE RESULT SET LOCATORS(result1, result2, result3)
      WITH PROCEDURE targetProcedure;
   ```

4. Asigne (ALLOCATE) los cursores de conjunto de resultados que se han pasado desde el procedimiento invocado a los localizadores de conjunto de resultados. Por ejemplo:

   ```
   ALLOCATE rsCur CURSOR FOR RESULT SET result1;
   ```

5. Capte (FETCH) las filas desde los conjuntos de resultados. Por ejemplo:

   ```
   FETCH rsCur INTO ...
   ```

Creación de procedimientos de SQL

La creación de procedimientos de SQL es parecida a la creación de cualquier objeto de base de datos en lo referente a la ejecución de una sentencia DDL de SQL.

Los procedimientos de SQL se crean ejecutando la sentencia CREATE PROCEDURE que se puede efectuar utilizando herramientas gráficas del entorno de desarrollo o ejecutando directamente la sentencia desde el Procesador de línea de mandatos (CLP) de DB2, una ventana de mandatos de DB2, el editor de mandatos de DB2 u otra interfaz de DB2.

Al crear procedimientos de SQL, puede especificar cómo el precompilador y el vinculador deben generar el paquete de procedimientos, qué ID de autorización se debe utilizar para establecer el definidor del procedimiento SQL en las vistas de catálogo de DB2, y establecer otras opciones del paquete.

Creación de procedimientos de SQL desde la línea de mandatos

Requisitos previos

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

Procedimiento

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

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

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

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

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

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

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

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

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

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

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

Personalización de las opciones de precompilación y vinculación para procedimientos de SQL

Las opciones de precompilación y enlace de los procedimientos de SQL se pueden personalizar estableciendo la variable de registro de DB2, DB2_SQLROUTINE_PREPOPTS con el mandato:

```
db2set DB2_SQLROUTINE_PREPOPTS=<opciones>
```

Sólo están permitidas las opciones siguientes:

```
BLOCKING {UNAMBIG | ALL | NO}
DATETIME {DEF | USA | EUR | ISO | JIS | LOC}
DEGREE {1 | grado-de-paralelismo | ANY}
DYNAMICRULES {BIND | RUN | DEFINERUN | DEFINEBIND | INVOKERUN | INVOKEBIND }
EXPLAIN {NO | YES | ALL}
EXPLSNAP {NO | YES | ALL}
FEDERATED {NO | YES}
INSERT {DEF | BUF}
ISOLATION {CS |RR |UR |RS |NC}
QUERYOPT nivel-optimización
REOPT {ALWAYS |NONE |ONCE}
VALIDATE {RUN | BIND}
```

Nota: El bloqueo del cursor está inhabilitado en los procedimientos SQL para los cursores a los que se hace referencia en sentencias FETCH y para los cursores implícitos en las sentencias FOR. Sin tener en cuenta el valor especificado para la opción de vinculación BLOCKING, se recuperarán datos de fila en fila de un modo optimizado altamente eficaz.
Estas opciones se pueden modificar a nivel de procedimiento mediante el procedimiento almacenado SET_ROUTINE_OPTS. Los valores de las opciones establecidos para crear procedimientos de SQL en la sesión actual se pueden obtener mediante la función GET_ROUTINE_OPTS.

Ejemplo.

Los procedimientos de SQL utilizados en este ejemplo se definirán en scripts del CLP (que se proporcionan a continuación). Estos scripts no están en el directorio de ejemplos sqlproc, pero los puede crear fácilmente cortando y pegando las sentencias de procedimiento CREATE en sus propios archivos.

En los ejemplos se utiliza una tabla denominada "expenses", que puede crear en la base de datos de ejemplo del modo siguiente:

```
db2 connect to sample
db2 CREATE TABLE expenses(amount DOUBLE, date DATE)
db2 connect reset
```

Para empezar, especifique el uso del formato ISO para las fechas como valor para toda la instancia:

```
db2set DB2_SQLROUTINE_PREPOPTS="DATETIME ISO"
db2stop
db2start
```

Es necesario detener y reiniciar DB2 para que el cambio entre en vigor.

A continuación, conectaremos con la base de datos:

```
db2 connect to sample
```

El primer procedimiento se define en el script del CLP maxamount.db2 del modo siguiente:

```
CREATE PROCEDURE maxamount(OUT maxamnt DOUBLE)
BEGIN
  SELECT max(amount) INTO maxamnt FROM expenses;
END @
```

Se creará con las opciones DATETIME ISO y ISOLATION UR:

```
db2 "CALL SET_ROUTINE_OPTS(GET_ROUTINE_OPTS() || ' ISOLATION UR')"
db2 -td@ -vf maxamount.db2
```

El siguiente procedimiento se define en el script del CLP fullamount.db2 del modo siguiente:

```
CREATE PROCEDURE fullamount(OUT fullamnt DOUBLE)
BEGIN
  SELECT sum(amount) INTO fullamnt FROM expenses;
END @
```

Se creará con la opción ISOLATION CS (observe que en este caso, no estamos utilizando el valor DATETIME ISO para toda la instancia):

```
CALL SET_ROUTINE_OPTS('ISOLATION CS')
db2 -td@ -vf fullamount.db2
```

El último procedimiento del ejemplo se define en el script del CLP perday.db2 del modo siguiente:

```
CREATE PROCEDURE perday()
BEGIN
  DECLARE cur1 CURSOR WITH RETURN FOR
    SELECT date, sum(amount)
    FROM expenses
    GROUP BY date;

    OPEN cur1;
END @
```

La última llamada a SET_ROUTINE_OPTS utiliza el valor NULL como argumento. Así se restaura el valor global especificado en el registro DB2_SQLROUTINE_PREPOPTS, de tal modo que el último procedimiento se creará con la opción DATETIME ISO:

```
CALL SET_ROUTINE_OPTS(NULL)
db2 -td@ -vf perday.db2
```

Mejora del rendimiento de los procedimientos de SQL

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

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

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

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

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

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

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

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

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

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

Aquí está la función de SQL reescrita:

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

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

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

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

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

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

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

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

se pueden reescribir como una sola sentencia:

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

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

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

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

se puede escribir como una sola sentencia VALUES:

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

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

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

La conversión de las dos sentencias anteriores a:

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

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

Reduzca varias sentencias de SQL a una sola expresión de SQL

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

Tome en consideración el siguiente fragmento de código SQL PL:

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

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

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

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

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

se puede reescribir como:

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

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

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

No subestime los beneficios que puede conseguir tomándose el tiempo necesario para analizar y valorar la posibilidad de volver a escribir la función de SQL. Los beneficios en el rendimiento le compensarán con creces el tiempo invertido en analizar y reescribir el procedimiento.

Aproveche la semántica de SQL establecida a la vez

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

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

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

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

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

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

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

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

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

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

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

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

Mantenga informado al optimizador de DB2

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

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

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

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

Utilización de matrices

Las matrices permiten pasar de forma eficiente colecciones de datos entre aplicaciones y procedimientos almacenados y almacenar y manipular colecciones

de datos transitorias entre procedimientos SQL sin tener que utilizar tablas relacionales. Los operadores de matrices disponibles en los procedimientos SQL permiten el almacenamiento y recuperación eficiente de los datos. Las aplicaciones que crean matrices de tamaño razonable obtendrán un rendimiento bastante mejor que otras que crean matrices muy grandes (en el rango de varios megabytes), ya que toda la matriz se almacena en la memoria principal. Consulte los *Enlaces relacionados* para obtener información adicional.

Capítulo 5. Visión general de las rutinas externas

Las rutinas externas se caracterizan principalmente por el hecho de que su lógica de rutina se implementa en código de lenguaje de programación y no en SQL.

Antes de decidir la implementación de una rutina externa, es importante que comprenda qué son las rutinas externas, cómo se implementan y cómo pueden utilizarse. Los temas sobre los conceptos siguientes lo ayudarán a comprender las rutinas externas para poder tomar decisiones informadas sobre cuándo y cómo utilizarlas en el entorno de bases de datos:

- "Funciones de rutinas externas"
- "Creación de rutinas externas" en la página 120
- "Gestión de bibliotecas y clases de rutinas externas" en la página 121
- "Interfaces API y lenguajes de programación soportados para el desarrollo de rutinas externas" en la página 23
- "Soporte de 32 bits y 64 bits para rutinas externas" en la página 131
- "Estilos de parámetros de rutinas externas" en la página 133
- "Restricciones para rutinas externas" en la página 135

Cuando haya comprendido los conceptos sobre rutinas externas, puede consultar:

- "Creación de rutinas externas" en la página 151

Rutinas externas

Las rutinas externas son rutinas cuya lógica está implementada en una aplicación de lenguaje de programación que reside fuera de la base de datos, en el sistema de archivos del servidor de bases de datos. La asociación de la rutina con la aplicación de código externo está indicada mediante la especificación de la cláusula EXTERNAL en la sentencia CREATE de la rutina.

Puede crear procedimientos externos, funciones externas y métodos externos. Aunque todos ellos se implementan en lenguajes de programación externos, cada tipo funcional de rutina tiene características distintas. Antes de decidir la implementación de una rutina externa, es importante que primero comprenda qué son las rutinas externas, cómo se implementan y se utilizan leyendo el tema "Visión general de las rutinas externas". Una vez comprenda esto, podrá obtener más información acerca de las rutinas externas en los temas de los enlaces relacionados para tomar decisiones informadas sobre cuándo y cómo utilizarlas en el entorno de bases de datos.

Funciones de rutinas externas

Las rutinas externas proporcionan soporte para las funciones más comunes de rutinas, así como soporte para funciones adicionales no soportadas por rutinas SQL. Las siguientes funciones son exclusivas de las rutinas externas:

Acceso a archivos, datos y aplicaciones que residen fuera de la base de datos
Las rutinas externas pueden acceder y manipular datos o archivos que residan fuera de la propia base de datos. También pueden invocar aplicaciones que residan fuera de la base de datos. Los datos, archivos o

© Copyright IBM Corp. 1993, 2010

aplicaciones podrían residir, por ejemplo, en el sistema de archivos del servidor de bases de datos o en la red disponible.

Variedad de opciones de estilos de parámetros de rutinas externas
La implementación de rutinas externas en un lenguaje de programación se puede realizar utilizando varias opciones de estilos de parámetros. Aunque puede haber un estilo de parámetro preferido para un determinado lenguaje de programación, a veces es posible escoger. Algunos estilos de parámetros proporcionan soporte para pasar información adicional sobre propiedades de rutinas y de bases de datos a la rutina y de recibirla de esta en una estructura denominada estructura *dbinfo* que puede resultar útil dentro de la lógica de la rutina.

Conservación de estado entre invocaciones de funciones externas con un área reutilizable
Las funciones externas definidas por el usuario proporcionan soporte para la conservación de estado entre invocaciones de función para un conjunto de valores. Esto se realiza con una estructura denominada *scratchpad*. Esto puede resultar útil tanto para las funciones que devuelven valores agregados como para las funciones que necesitan lógica de configuración inicial, como inicialización de almacenamientos intermedios.

Los tipos de llamada identifican invocaciones de funciones externas individuales
Las funciones externas definidas por el usuario se invocan varias veces para un conjunto de valores. Cada invocación se identifica con un valor de tipo de llamada al que se puede hacer referencia dentro de la lógica de la función. Por ejemplo, hay tipos de llamada especiales para la primera invocación de una función, para las llamadas de captación de datos y para la invocación final. Los tipos de llamada resultan útiles porque se puede asociar lógica específica a un tipo de llamada determinado.

Creación de rutinas externas

Las rutinas externas se crean de manera parecida a cómo se crean rutinas con otras implementaciones. Sin embargo, se necesitan algunos pasos adicionales porque, para implementar la rutina, hay que escribir código fuente, compilarlo y desplegarlo.

Una rutina externa consta de dos partes:

- La sentencia CREATE que define la rutina.
- La biblioteca o clase externa que implementa el cuerpo de la rutina.

Cuando la sentencia CREATE que define una rutina se ejecuta satisfactoriamente, la rutina se crea en la base de datos. La sentencia debe definir como mínimo el nombre de la rutina, la signatura de parámetros que se usará en la implementación de la rutina, y la ubicación de la biblioteca o clase externa construida a partir del código fuente de la implementación de la rutina.

El código de implementación de las rutinas externas debe estar escrito en uno de los lenguajes de programación soportados y luego hay que incorporarlo en un archivo de clase o biblioteca que debe estar instalado en el sistema de archivos del servidor de bases de datos.

Una rutina externa no se puede invocar satisfactoriamente hasta que se ha creado en la base de datos y hasta que la biblioteca o clase asociada a la rutina se ha colocado en la ubicación especificada por la cláusula EXTERNAL.

El desarrollo de rutinas externas suele consistir en las siguientes tareas:

- Determinar qué tipo funcional de rutina hay que implementar.
- Elegir uno de los lenguajes de programación soportados para las rutinas externas de cara a su implementación.
- Diseñar la rutina.
- Conectar con una base de datos y crear la rutina en la base de datos.
  - Para ello, se ejecuta una de las sentencias CREATE PROCEDURE, CREATE FUNCTION o CREATE METHOD o se utiliza una herramienta gráfica que automatice este paso.
  - Esta tarea, también conocida como definición o registro de una rutina, se puede realizar en cualquier momento antes de invocar la rutina, excepto en las circunstancias siguientes:
    - Para las rutinas Java que hacen referencia a un archivo u archivos JAR externos, el código y los archivos JAR externos se deben codificar y compilar antes de que se cree la rutina en la base de datos utilizando la sentencia CREATE específica del tipo de rutina.
    - Las rutinas que ejecutan sentencias de SQL y se refieren directamente a sí mismas se deben crear en la base de datos emitiendo la sentencia CREATE antes de que se precompile y enlace el código externo asociado con la rutina. Esto también es aplicable a las situaciones en que exista un ciclo de referencias; por ejemplo: la Rutina A hace referencia a la Rutina B, la cual hace referencia a la Rutina A.
- Escribir el código de la lógica de la rutina para que se corresponda con la definición de la rutina.
- Construir la rutina y generar un archivo de clase o biblioteca.
  - En el caso de las rutinas de SQL incorporado, incluye: precompilar, compilar y enlazar el código, así como enlazar el paquete de la rutina con la base de datos destino.
  - En el caso de las rutinas de SQL no incorporado, incluye: compilar y enlazar el código.
- Desplegar el archivo de clase o biblioteca en el servidor de bases de datos, en la ubicación especificada en la definición de la rutina.
- Otorgar el privilegio EXECUTE sobre la rutina al invocador o invocadores de la rutina (si no son los que han definido la rutina).
- Invocar, probar y depurar la rutina.

Los pasos necesarios para crear una rutina externa se pueden realizar todos mediante el procesador de línea de mandatos de DB2 o una ventana de mandatos de DB2. Hay herramientas que pueden ayudar a automatizar algunos de estos pasos o todos ellos.

Gestión de bibliotecas y clases de rutinas externas

Para desarrollar e invocar rutinas externas satisfactoriamente, los archivos de bibliotecas y de clases de rutinas externas deben desplegarse y gestionarse debidamente.

La gestión de archivos de bibliotecas y de clases de rutinas externas puede ser mínima si se tiene cuidado cuando se crean las rutinas externas por primera vez y se despliegan archivos de bibliotecas y clases de rutinas externas.

Las principales consideraciones a tener en cuenta sobre la gestión de rutinas externas son las siguientes:

- Despliegue de archivos de bibliotecas y de clases de rutinas externas
- Seguridad de archivos de bibliotecas y de clases de rutinas externas
- Resolución de bibliotecas y clases de rutinas externas
- Modificaciones de archivos de bibliotecas y de clases de rutinas externas
- Copia de seguridad y restauración de archivos de bibliotecas y de clases de rutinas externas

Los administradores del sistema, los administradores de bases de datos y los desarrolladores de aplicaciones de bases de datos deben responsabilizarse de garantizar que los archivos de bibliotecas y de clases de las rutinas externas están protegidos y correctamente conservados durante las tareas de desarrollo de rutinas y de administración de bases de datos.

Despliegue de bibliotecas y clases de rutinas externas

El despliegue de bibliotecas y clases de rutinas externas consiste en copiar las bibliotecas y clases de las rutinas externas en el servidor de bases de datos cuando se han creado a partir del código fuente.

Los archivos de bibliotecas de rutinas externas, clases o ensamblajes deben copiarse en el directorio *function* de DB2 o en un subdirectorio de este directorio en el servidor de bases de datos. Esta es la ubicación recomendada para el despliegue de rutinas externas. Para obtener más información sobre el directorio de función, consulte la descripción de la cláusula EXTERNAL correspondiente a cualquiera de las siguientes sentencias de SQL: CREATE PROCEDURE o CREATE FUNCTION.

Puede copiar la clase, biblioteca o ensamblaje de la rutina externa en otras ubicaciones de directorios del servidor, en función de la API y el lenguaje de programación utilizados para implementar la rutina, aunque no se recomienda hacerlo en general. Si se hace, para invocar satisfactoriamente la rutina debe anotar el nombre de la vía de acceso calificada al completo y asegurarse de que se utilice este valor con la cláusula EXTERNAL NAME.

Los archivos de biblioteca y de clase se pueden copiar en el sistema de archivos del servidor de bases de datos mediante las herramientas de transferencia de archivos disponibles de manera más general. Las rutinas Java se pueden copiar desde un sistema donde hay un cliente de DB2 instalado en un servidor de bases de datos de DB2 mediante procedimientos especiales definidos por el sistema diseñados específicamente para ello. Para más información, consulte el tema sobre rutinas Java.

Cuando ejecute la sentencia CREATE del lenguaje SQL adecuado correspondiente al tipo de rutina, CREATE PROCEDURE o CREATE FUNCTION, asegúrese de especificar las cláusulas adecuadas, prestando especial atención en la cláusula EXTERNAL NAME.

- Especifique la cláusula `LANGUAGE` con el valor adecuado correspondiente a la API o lenguaje de programación elegido. Ejemplos: CLR, C, JAVA.
- Especifique la cláusula `PARAMETER STYLE` con el nombre del estilo de parámetro soportado que se ha implementado en el código de la rutina.
- Especifique la cláusula `EXTERNAL` con el nombre del archivo de biblioteca, clase o ensamblaje que se ha de asociar a la rutina utilizando uno de los valores siguientes:

- el nombre de vía de acceso calificado al completo del archivo de biblioteca, clase o ensamblaje de la rutina.
- el nombre de vía de acceso relativa del archivo de biblioteca, clase o ensamblaje de la rutina con relación al directorio de función.

Por omisión, DB2 buscará el archivo de biblioteca, clase o ensamblaje por el nombre en el directorio de función, a menos que se especifique un nombre de vía de acceso completamente calificado o relativo para el archivo en la cláusula EXTERNAL.

Seguridad de archivos de bibliotecas o clases de rutinas externas

Las bibliotecas de las rutinas externas se almacenan en el sistema de archivos en el servidor de bases de datos, y el gestor de bases de datos de DB2 no hace copia de ellas ni las protege de ningún modo. Para que se pueda seguir invocando satisfactoriamente las rutinas, es imprescindible que la biblioteca asociada a la rutina continúe existiendo en la ubicación especificada en la cláusula EXTERNAL de la sentencia CREATE utilizada para crear la rutina. No mueva ni suprima bibliotecas de rutinas después de crear las rutinas; de lo contrario, las invocaciones de las rutinas fallarían.

Para evitar que las bibliotecas de rutinas se supriman o sustituyan de forma accidental o intencionada, debe restringir el acceso a los directorios del servidor de bases de datos que contienen bibliotecas de rutinas y restringir el acceso a los archivos de bibliotecas de rutinas. Esto se puede realizar utilizando mandatos del sistema operativo para establecer permisos sobre directorios y archivos.

Resolución de bibliotecas y clases de rutinas externas

La resolución de bibliotecas de rutinas externas de DB2 se realiza a nivel de instancia de DB2. Esto significa que, en instancias de DB2 que contienen varias bases de datos de DB2, se pueden crear rutinas externas en una base de datos que utilicen bibliotecas de rutinas externas que ya se utilicen para una rutina en otra base de datos.

La resolución de rutinas externas a nivel de instancia da soporte a la reutilización de código, permitiendo asociar varias definiciones de rutina a una sola biblioteca. Cuando las bibliotecas de rutinas externas no se reutilizan de esta forma, sino que existen copias de la biblioteca de la rutina externa en el sistema de archivos del servidor de bases de datos, pueden producirse conflictos de nombres de bibliotecas. Esto puede suceder concretamente cuando hay varias bases de datos en una sola instancia y las rutinas de cada base de datos se asocian a sus propias copias de bibliotecas y clases de cuerpos de rutinas. Se produce un conflicto cuando el nombre de una biblioteca o de una clase utilizado por una rutina de una base de datos es idéntico al nombre de la biblioteca o clase utilizado por una rutina de otra base de datos (de la misma instancia).

Para minimizar la probabilidad de que esto suceda, se recomienda almacenar una sola copia de una biblioteca de rutina en el directorio de función de nivel de instancia (directorio sqllib/function) y que la cláusula EXTERNAL de todas las definiciones de rutinas de cada una de las bases de datos haga referencia a la biblioteca exclusiva.

Si se tienen que crear dos bibliotecas de rutinas funcionalmente diferentes con el mismo nombre, es importante realizar pasos adicionales para minimizar la probabilidad de que se produzcan conflictos de nombres de bibliotecas.

Para rutinas C, C++, COBOL y ADO.NET:

Los conflictos de nombres de bibliotecas se pueden minimizar o resolver:

1. Almacenando las bibliotecas con los cuerpos de las rutinas en distintos directorios para cada base de datos.
2. Creando las rutinas con un valor de cláusula EXTERNAL NAME que especifique la vía de acceso completa de una determinada biblioteca (en lugar de una vía de acceso relativa).

Para las rutinas de Java:

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

Modificaciones de archivos de bibliotecas y de clases de rutinas externas

Puede ser necesario modificar la lógica de una rutina externa existente después de que una rutina externa se despliegue y se utilice en un entorno del sistema de bases de datos de producción. Se pueden realizar modificaciones en las rutinas existentes, pero hay que realizarlas cuidadosamente para definir un momento de toma de control claro para las actualizaciones y para minimizar el riesgo de que se produzcan interrupciones en cualquier invocación concurrente de la rutina.

Si una biblioteca de rutina externa necesita una actualización, no vuelva a compilar y a enlazar la rutina con el mismo archivo de destino (por ejemplo, `sqllib/function/foo.a`) que utiliza la rutina actual mientras se está ejecutando el gestor de bases de datos. Si una invocación de rutina actual accede a una versión en antememoria del proceso de la rutina y se sustituye la biblioteca subyacente, la invocación de la rutina puede fallar. Si hay que modificar el cuerpo de una rutina sin detener y volver a iniciar DB2, siga los pasos siguientes:

1. Cree la nueva biblioteca de rutina externa con otro nombre de archivo de biblioteca o de clase.
2. Si se trata de una rutina de SQL incorporado, enlace el paquete de la rutina a la base de datos con el mandato BIND.
3. Utilice la sentencia ALTER ROUTINE para cambiar la definición de la rutina para que la cláusula EXTERNAL NAME haga referencia a la biblioteca o clase de la rutina actualizada. Si el cuerpo de rutina que se debe actualizar lo utilizan rutinas catalogadas en varias bases de datos, las acciones recomendadas en esta sección se deberán llevar a cabo para cada base de datos afectada.
4. Para actualizar rutinas Java incorporadas en archivos JAR, tiene que emitir una sentencia CALL SQLJ.REFRESH_CLASSES() a fin de forzar que DB2 cargue las nuevas clases. Si no emite la sentencia CALL SQLJ.REFRESH_CLASSES() después de actualizar las clases de rutinas Java, DB2 continúa utilizando las versiones anteriores de las clases. DB2 renueva las clases cuando se produce una operación COMMIT o ROLLBACK.

Una vez actualizada la definición de la rutina, las siguientes invocaciones de la rutina cargarán y ejecutarán la nueva biblioteca o clase de rutina externa.

Copia de seguridad y restauración de archivos de bibliotecas y de clases de rutinas externas

No se hace copia de seguridad de las bibliotecas de rutinas externas con otros objetos de base de datos cuando se realiza una copia de seguridad de la base de datos. Tampoco se restauran cuando se restaura una base de datos.

Si el objetivo de una operación de copia de seguridad y restauración de una base de datos es redesplegar una base de datos, los archivos de bibliotecas de rutinas externas del sistema de archivos del servidor de bases de datos original se tienen que copiar en el sistema de archivos del servidor de bases de datos de destino, de forma que se conserven los nombres de vías de acceso relativas de las bibliotecas de rutinas externas.

Gestión y rendimiento de bibliotecas de rutinas externas

La gestión de bibliotecas de rutinas externas puede afectar al rendimiento de las rutinas, puesto que el gestor de bases de datos de DB2 coloca en antememoria de forma dinámica las bibliotecas de rutinas externas a fin de mejorar el rendimiento de acuerdo con el uso de las rutinas. para conseguir un rendimiento óptimo de las rutinas externas, tenga en cuenta lo siguiente:

- Haga que el número de rutinas de cada biblioteca sea lo más reducido posible. Es mejor tener muchas bibliotecas de rutinas externas pequeñas que unas pocas de gran tamaño.
- Agrupe dentro del código fuente las funciones de las rutinas que se suelen invocar juntas. Cuando el código se compila en una biblioteca de rutinas externas, los puntos de entrada de las rutinas que se invocan con frecuencia estarán juntos, lo que permite al gestor de bases de datos ofrecer un mejor soporte de colocación en antememoria. El soporte mejorado de colocación en antememoria se debe a la eficacia que se puede obtener al cargar una sola biblioteca de rutinas externas una vez y luego invocar varias funciones de rutinas externas dentro de dicha biblioteca.

  Para las rutinas externas implementadas en el lenguaje de programación C o C++, el coste de cargar una biblioteca se paga una vez para las bibliotecas que se encuentran en uso de forma coherente por parte de las rutinas C. Después de la primera invocación de la rutina, no es necesario que todas las invocaciones posteriores, de la misma hebra del proceso, vuelvan a cargar la biblioteca de la rutina.

Interfaces API y lenguajes de programación soportados para el desarrollo de rutinas externas

Puede desarrollar rutinas externas de DB2 (procedimientos y funciones) utilizando las siguientes API y los lenguajes de programación asociados:

- ADO.NET
  - Lenguajes de programación .NET Common Language Runtime
- CLI
- SQL incorporado
  - C
  - C++

  - COBOL (solo para procedimientos)
- JDBC
  - Java
- OLE
  - Visual Basic
  - Visual C++
  - Cualquier otro lenguaje de programación que soporte esta API.
- OLE DB (solo para funciones de tabla)
  - Cualquier lenguaje de programación que soporte esta API.
- SQLJ
  - Java

Comparación de las interfaces API y los lenguajes de programación soportados para el desarrollo de rutinas externas

Antes de empezar a implementar rutinas externas, es importante tener en cuenta las características y las limitaciones de las distintas interfaces de programación de aplicaciones (API) y lenguajes de programación soportados para las rutinas externas. Con ello se asegurará de que elige la implementación adecuada desde el principio y de que están disponibles las características de rutinas que necesite.

Tabla 10. Comparación de las API y lenguajes de programación de rutinas externas

API y lenguaje de programación	Soporte de características	Rendimiento	Seguridad	Escalabilidad	Limitaciones
SQL (incluye SQL PL)	• SQL es un lenguaje de alto nivel fácil de aprender y de utilizar y que agiliza la implementación. • Los elementos del Lenguaje de procedimientos de SQL (SQL PL) permiten utilizar la lógica de flujo de control en operaciones y consultas SQL. • Soporte a tipos de datos firmes.	• Muy bueno. • El rendimiento de las rutinas SQL es mejor que el de las rutinas Java. • El rendimiento de las rutinas SQL es tan bueno como el de las rutinas externas de C y C++ creadas con la cláusula NOT FENCED.	• Muy seguro. • Los procedimientos de SQL siempre se ejecutan en la misma memoria que el gestor de bases de datos. Esto corresponde a la rutina que se está creando de forma predeterminada con las palabras clave NOT FENCED.	• Gran escalabilidad.	• No pueden acceder al sistema de archivos del servidor de bases de datos. • No pueden invocar aplicaciones que residan fuera de la base de datos.

Tabla 10. Comparación de las API y lenguajes de programación de rutinas externas (continuación)

API y lenguaje de programación	Soporte de características	Rendimiento	Seguridad	Escalabilidad	Limitaciones
SQL incorporado (incluye C y C++)	• Lenguaje de programación de bajo nivel, pero potente.	• Muy bueno. • El rendimiento de las rutinas de C y C++ es mejor que el de las rutinas Java. • El rendimiento de las rutinas de C y C++ creadas con la cláusula NOT FENCED es tan bueno como el de las rutinas SQL.	• Las rutinas de C y C++ son susceptibles a los errores de programación. • Los programadores deben tener un buen dominio del lenguaje C para evitar cometer errores de memoria y de manipulación de punteros muy comunes que hacen que la implementación de las rutinas sea más pesada y requiera más tiempo. • Las rutinas de C y C++ deben crearse con la cláusula FENCED y la cláusula NOT THREADSAFE para evitar la interrupción del gestor de bases de datos en caso de que se produzca una excepción en la rutina en tiempo de ejecución. Éstas son las cláusulas por omisión. La utilización de estas cláusulas puede tener un cierto efecto negativo sobre el rendimiento pero garantiza una ejecución segura. Consulte: Seguridad de las rutinas.	• La escalabilidad es limitada cuando se crean rutinas de C y C++ con las cláusulas FENCED y NOT THREADSAFE. Estas rutinas se ejecutan en un proceso *db2fmp* independiente del proceso del gestor de bases de datos. Se necesita un proceso *db2fmp* para cada rutina que se ejecute de forma simultánea.	• Existen múltiples estilos para pasar los parámetros soportados y esto puede resultar confuso. Los usuarios deben utilizar siempre que sea posible el estilo de los parámetros SQL.

Tabla 10. Comparación de las API y lenguajes de programación de rutinas externas (continuación)

API y lenguaje de programación	Soporte de características	Rendimiento	Seguridad	Escalabilidad	Limitaciones
SQL incorporado (COBOL)	• Lenguaje de programación de alto nivel adecuado para el desarrollo de aplicaciones empresariales que suelen ir orientadas a archivos. • Utilizado de forma generalizada en el pasado para las aplicaciones empresariales de producción, aunque su popularidad está decayendo. • COBOL no contiene soporte a los punteros y es un lenguaje de programación iterativo y lineal.	• El rendimiento de las rutinas de COBOL no es tan bueno como el de las rutinas creadas con el resto de opciones de implementación de rutinas externas.	• No hay información disponible en estos momentos.	• No hay información disponible en estos momentos.	• Es posible crear e invocar procedimientos de COBOL de 32 bits en instancias de DB2 de 64 bits; sin embargo, el rendimiento de estas rutinas no será tan bueno como el de los procedimientos de COBOL de 64 bits de una instancia de DB2 de 64 bits.
JDBC (Java) y SQLJ (Java)	• Lenguaje de programación de alto nivel orientado a objetos adecuado para el desarrollo de aplicaciones autónomas, applets y servlets. • Los objetos y los tipos de datos de Java facilitan el establecimiento de conexiones con la base de datos, la ejecución de las sentencias de SQL y la manipulación de los datos.	• El rendimiento de las rutinas de Java es tan bueno como el de las rutinas de C y C++ o el de las rutinas SQL.	• Las rutinas de Java son más seguras que las rutinas de C y C++ porque la Máquina virtual Java (JVM) es quien gestiona el control de las operaciones peligrosas. Esto aumenta la fiabilidad y hace que resulte difícil que el código de una rutina de Java perjudique a otra rutina que se ejecute en el mismo proceso.	• Buena escalabilidad • Las rutinas Java creadas con la cláusula FENCED THREADSAFE (el valor por omisión) tienen una buena escalabilidad. Todas las rutinas Java protegidas comparten algunas JVM. Es posible que se utilice más de una JVM en el sistema si la pila de Java de un proceso db2fmp concreto está próxima a su agotamiento.	• Para evitar operaciones potencialmente peligrosas, no se permiten llamadas JNI (Java Native Interface) desde las rutinas Java.

Tabla 10. Comparación de las API y lenguajes de programación de rutinas externas (continuación)

API y lenguaje de programación	Soporte de características	Rendimiento	Seguridad	Escalabilidad	Limitaciones
Lenguajes soportados Common Language Runtime (CLR) .NET (incluidos C#, Visual Basic y otros)	• Parte del modelo de código gestionado .NET de Microsoft. • El código fuente se compila en el bytecode de lenguaje intermedio (IL) que se puede interpretar mediante la unidad ejecutable de lenguaje común (CLR) Microsoft .NET Framework. • Los ensamblajes CLR se pueden crear a partir de subensamblajes que se hayan compilado con diferente código fuente de lenguaje de programación .NET, lo que permite a los usuarios reutilizar e integrar módulos de código escritos en varios lenguajes.	Cuando se crean rutinas que especifican la cláusula LANGUAGE CLR, se debe especificar también la cláusula FENCED. La cláusula FENCED se utiliza por omisión. La cláusula THREADSAFE también se utiliza por omisión, pero se puede especificar la cláusula NOT THREADSAFE. La utilización de los valores de cláusula por omisión minimiza la interrupción del gestor de bases de datos durante la ejecución, pero debido a que las rutinas CLR se deben ejecutar como FENCED, su rendimiento puede ser ligeramente menor que el de rutinas externas con otras implemen-taciones que se pueden especificar como NOT FENCED.	• Cuando se crean rutinas que especifican la cláusula LANGUAGE CLR, se debe especificar también la cláusula FENCED. La cláusula FENCED se utiliza por omisión. La cláusula THREADSAFE también se utiliza por omisión, pero se puede especificar la cláusula NOT THREADSAFE. Estas rutinas son seguras porque se ejecutan fuera del gestor de bases de datos en un proceso db2fmp separado.	• No hay información disponible.	Consulte el tema "Restricciones de las rutinas CLR de .NET".

Tabla 10. Comparación de las API y lenguajes de programación de rutinas externas (continuación)

API y lenguaje de programación	Soporte de características	Rendimiento	Seguridad	Escalabilidad	Limitaciones
• OLE	• Las rutinas de OLE se pueden implantar en Visual C++, en Visual Basic y en otros lenguajes soportados por OLE.	• La velocidad de las rutinas automatizadas OLE dependerá del lenguaje utilizado para implementarlas. En general, son más lentas que las rutinas C/C++ que no son OLE. • Las rutinas OLE solo se pueden ejecutar en modalidad FENCED NOT THREADSAFE y, por lo tanto, las rutinas automatizadas OLE tienen una buena escalabilidad.	• No hay información disponible.	• No hay información disponible.	• No hay información disponible.

Tabla 10. Comparación de las API y lenguajes de programación de rutinas externas (continuación)

API y lenguaje de programación	Soporte de características	Rendimiento	Seguridad	Escalabilidad	Limitaciones
• OLE DB	• OLE DB se puede utilizar para crear funciones de tabla definidas por el usuario. • Las funciones OLE DB establecen conexión con fuentes de datos externas OLE DB.	• El rendimiento de las funciones OLE DB depende del proveedor de OLE DB; sin embargo, en general las funciones OLE DB ofrecen un mejor rendimiento que las funciones Java equivalente en términos lógicos pero menor que las funciones C, C++ o SQL equivalentes en términos lógicos. No obstante, algunos predicados de la consulta en que se invoca la función se pueden evaluar en el proveedor de OLE DB, por lo que quedará reducido el número de filas que DB2 tenga que procesar, lo que suele dar lugar a un mejor rendimiento.	• No hay información disponible.	• No hay información disponible.	• OLE DB sólo se puede utilizar para crear funciones de tabla definidas por el usuario.

Soporte de 32 bits y 64 bits para rutinas externas

El soporte para rutinas externas de 32 bits y 64 bits está determinado por la especificación de una de las dos cláusulas siguientes en la sentencia CREATE para las rutinas: cláusula FENCED o cláusula NOT FENCED.

El cuerpo de la rutina de una rutina externa se escribe en un lenguaje de programación y se compila en una biblioteca o archivo de clase que luego se carga y se ejecuta cuando se invoca la rutina. La especificación de la cláusula FENCED o NOT FENCED determina si la rutina externa se ejecuta en un entorno con barrera diferenciado del gestor de bases de datos o en el mismo espacio de direcciones que

el gestor de bases de datos, lo que puede ofrecer un mejor rendimiento mediante el uso de memoria compartida en vez de TCPIP para las comunicaciones. Por omisión, las rutinas siempre se crean con barrera independientemente de las otras cláusulas seleccionadas.

La tabla siguiente ilustra el soporte de DB2 para ejecutar rutinas de 32 bits y 64 bits con y sin barrera en servidores de base de datos de 32 bits y 64 bits que ejecutan el mismo sistema operativo.

Tabla 11. Soporte para rutinas externas de 32 y de 64 bits

Ancho de bits de la rutina	Servidor de 32 bits	Servidor de 64 bits
Procedimiento con barrera de 32 bits o UDF	Soportado	Soportado
Procedimiento con barrera de 64 bits o UDF	No soportado (4)	Soportado
Procedimiento sin barrera de 32 bits o UDF	Soportado	Soportado (2)
Procedimiento sin barrera de 64 bits o UDF	No soportado (4)	Soportado

Las notas a pie de página de la tabla anterior corresponden a:

- (1) Ejecutar una rutina de 32 bits en un servidor de 64 bits no es tan rápido como ejecutar una rutina de 64 bits en un servidor de 64 bits.
- (2) Las rutinas de 32 bits deben crearse como FENCED y NOT THREADSAFE para funcionar en un servidor de 64 bits.
- (3) No es posible invocar rutinas de 32 bits en servidores de base de datos de 64 bits Linux IA.
- (4) Las aplicaciones y las rutinas de 64 bits y no pueden ejecutarse en espacios de direcciones de 32 bits.

Lo importante a tener en cuenta en la tabla es que los procedimientos sin barrera de 32 bits no pueden ejecutarse en un servidor DB2. Si debe desplegar rutinas sin barrera de 32 bits en plataformas de 64 bits, elimine la cláusula NOT FENCED de las sentencias CREATE para estas rutinas antes de catalogarlas.

Rendimiento de las rutinas con bibliotecas de 32 bits en servidores de bases de datos de 64 bits

Es posible invocar rutinas con bibliotecas de rutinas de 32 bits en servidores de bases de datos de DB2. Sin embargo, su rendimiento no es tan bueno como el de invocar una rutina de 64 bits en un servidor de 64 bits. La razón de la disminución del rendimiento es que, antes de cada intento de ejecutar una rutina de 32 bits en un servidor de 64 bits, primero se intenta invocarla como una biblioteca de 64 bits. Si esto falla, entonces la biblioteca se invocará como una biblioteca de 32 bits. Un intento fallido de invocar una biblioteca de 32 bits como si fuera una biblioteca de 64 bits genera un mensaje de error (SQLCODE -444) en el archivo db2diag.log.

Las clases de Java son independientes del número de bits. Solamente se clasifican las máquinas virtuales Java de 32 ó 64 bits. DB2 sólo da soporte al uso de máquinas JVM que tengan la misma anchura que la instancia en la que se utilizan. En otras palabras, en una instancia de 32 bits DB2, sólo se puede utilizar una JVM

de 32 bits, y en una instancia de DB2 de 64 bits, sólo se puede utilizar una JVM de 64 bits. De este modo se garantiza el correcto funcionamiento de las rutinas Java y el mejor rendimiento posible.

Estilos de parámetros de rutinas externas

Las implementaciones de rutinas externas se tienen que ajustar a un convenio determinado para el intercambio de valores de parámetros de la rutina. Estos convenios se conocen como *estilos de parámetros*. Se especifica un estilo de parámetro de rutina externa cuando se crea la rutina, especificando la cláusula PARAMETER STYLE. Los estilos de parámetros caracterizan la especificación y el orden en el que se pasarán los valores de los parámetros a la implementación de la rutina externa. También especifican qué pasará si se pasan valores adicionales a la implementación de la rutina externa. Por ejemplo, algunos estilos de parámetros especifican que para cada valor de parámetro de rutina se debe pasar un valor de indicador de nulo adicional por separado a la implementación de la rutina para proporcionar información sobre la capacidad de nulos de los parámetros, lo que de lo contrario no se puede determinar fácilmente con un tipo de datos del lenguaje de programación nativo.

La tabla siguiente proporciona una lista de los estilos de parámetros disponibles, las implementaciones de rutina que dan soporte a cada estilo de parámetro, los tipos de rutinas funcionales que dan soporte a cada estilo de parámetro y una descripción del estilo de parámetro:

Tabla 12. Estilos de parámetros

Estilo de parámetros	Lenguaje soportado	Tipo de rutina soportado	Descripción
SQL [1]	• C/C++ • OLE • Lenguajes CLR (Commmon Language Runtime) de .NET • COBOL [2]	• UDF • procedimientos almacenados • métodos	Además de los parámetros que se pasan durante la invocación, se pasan a la rutina los argumentos siguientes por este orden: • Un indicador nulo para cada parámetro o resultado declarado en la sentencia CREATE. • El SQLSTATE que se debe devolver a DB2. • El nombre calificado de la rutina. • El nombre específico de la rutina. • La serie de diagnóstico de SQL que se debe devolver a DB2. Según las opciones especificadas en la sentencia CREATE y el tipo de rutina, se pueden pasar a la rutina los argumentos siguientes por este orden: • Un almacenamiento intermedio para el área reutilizable. • El tipo de llamada de la rutina. • La estructura dbinfo (contiene información acerca de la base de datos).

Tabla 12. Estilos de parámetros (continuación)

Estilo de parámetros	Lenguaje soportado	Tipo de rutina soportado	Descripción
DB2SQL [1]	• C/C++ • OLE • Lenguajes CLR (Commmon Language Runtime) de .NET • COBOL	• procedimientos almacenados	Además de los parámetros que se pasan durante la invocación, se pasan al procedimiento almacenado los argumentos siguientes por este orden: • Un vector que contiene un indicador nulo para cada parámetro de la sentencia CALL. • El SQLSTATE que se debe devolver a DB2. • El nombre calificado del procedimiento almacenado. • El nombre específico del procedimiento almacenado. • La serie de diagnóstico de SQL que se debe devolver a DB2. Si se especifica la cláusula DBINFO en la sentencia CREATE PROCEDURE, se pasa al procedimiento almacenado una estructura dbinfo (contiene información acerca de la base de datos).
JAVA	• Java	• UDF • procedimientos almacenados	Las rutinas con PARAMETER STYLE JAVA utilizan un convenio de pase de parámetros que cumple con la especificación del lenguaje Java y las rutinas de SQLJ. Para los procedimientos almacenados, los parámetros INOUT y OUT se pasarán como matrices de entrada única para facilitar la devolución de valores. Además de los parámetros IN, OUT e INOUT, las signaturas de método Java de los procedimientos almacenados incluyen un parámetro del tipo ResultSet[] para cada conjunto de resultados especificado en la cláusula DYNAMIC RESULT SETS de la sentencia CREATE PROCEDURE. Para las UDF y los métodos con PARAMETER STYLE JAVA, no se pasan más argumentos que los especificados en la invocación de la rutina. Las rutinas PARAMETER STYLE JAVA no proporcionan soporte a las cláusulas DBINFO o PROGRAM TYPE. Para las UDF, PARAMETER STYLE JAVA sólo puede especificarse si no se han especificado como parámetros tipos de datos estructurados y no se ha especificado como tipo de retorno ningún tipo de datos estructurado, CLOB, DBCLOB ni BLOB (SQLSTATE 429B8). Además las UDF de PARAMETER STYLE JAVA no proporcionan soporte a las funciones de tabla, los tipos de llamada ni áreas reutilizables.
DB2GENERAL	• Java	• UDF • procedimientos almacenados • métodos	Este tipo de rutina utilizará un convenio de pase de parámetros definido para su uso con los métodos Java. A no ser que se desarrollen UDF de tabla o UDF con áreas reutilizables o que se tenga que acceder a la estructura dbinfo, es recomendable utilizar PARAMETER STYLE JAVA. Para las rutinas con PARAMETER STYLE DB2GENERAL, no se pasa ningún argumento adicional aparte de los especificados en la invocación de la rutina.

Tabla 12. Estilos de parámetros (continuación)

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

Nota:

1. Para las UDF y los métodos, PARAMETER STYLE SQL es equivalente a PARAMETER STYLE DB2SQL.
2. COBOL sólo se puede utilizar para desarrollar procedimientos almacenados.
3. Los métodos de ejecución CLR (Common Language Runtime) .NET no están soportados.

Restricciones para rutinas externas

Se aplican las siguientes restricciones a las rutinas externas y deben tenerse en cuenta a la hora de desarrollar o depurar rutinas externas:

Restricciones que se aplican a todas las rutinas externas:

- En las rutinas externas no se pueden crear hebras nuevas.
- Las API de nivel de conexión no se pueden llamar desde dentro de funciones o métodos externos.
- Desde rutinas externas no es posible recibir entradas desde el teclado ni visualizar salidas en la salida estándar. No utilice corrientes de entrada-salida estándar. Por ejemplo:
  - En código de rutinas Java externas, no emita los métodos `System.out.println()`.
  - En código de rutinas C o C++ externas, no emita `printf()`.
  - En código de rutinas COBOL externas, no emita `display`

  Aunque las rutinas externas no pueden visualizar datos en una salida estándar, sí pueden incluir código que grabe datos en un archivo del sistema de archivos del servidor de bases de datos.

  Para las rutinas delimitadas (fenced) que se ejecutan en entornos UNIX, el directorio de destino en que se debe crear el archivo, o el propio archivo, debe tener los permisos adecuados, de forma que el propietario del archivo

sqllib/adm/.fenced pueda crearlo o grabar en él. Si se trata de rutinas no delimitadas (not fenced), el propietario de la instancia debe tener permisos de creación, lectura y grabación para el directorio en que se abra el archivo.

Nota: DB2 no intenta sincronizar ninguna entrada ni salida externa que una rutina con transacciones propias de DB2 realice. Así, por ejemplo, si una UDF graba en un archivo durante una transacción y más tarde se retira dicha transacción por cualquier motivo, no se realiza ningún intento de descubrir ni deshacer las grabaciones efectuadas en el archivo.

- En rutinas externas no pueden ejecutarse sentencias o mandatos relacionados con la conexión. Esta restricción es aplicable a las siguientes sentencias:
  - BACKUP
  - CONNECT
  - CONNECT TO
  - CONNECT RESET
  - CREATE DATABASE
  - DROP DATABASE
  - FORWARD RECOVERY
  - RESTORE
- No es recomendable utilizar funciones del sistema operativo dentro de las rutinas. El uso de estas funciones no está restringido, excepto en los casos siguientes:
  - **No deben instalarse manejadores de señales definidos por el usuario para rutinas externas. Si no se cumple con esta restricción, se pueden producir anomalías inesperadas durante la ejecución de las rutinas externas, terminaciones anormales de base de datos u otros problemas. La instalación de manejadores de señales también puede interferir en el funcionamiento de la JVM para las rutinas de Java.**
  - Las llamadas al sistema que terminan un proceso pueden terminar de forma anómala uno de los procesos de DB2 y causar una anomalía del sistema o de la aplicación de base de datos.

    Otras llamadas al sistema también pueden ocasionar problemas si interfieren en el funcionamiento normal del gestor de bases de datos DB2. Por ejemplo, una función que intente descargar una biblioteca que contenga una función definida por el usuario desde la memoria podría causar problemas graves. Preste atención al programar y al probar las rutinas externas que contengan llamadas al sistema.
- Las rutinas externas no deben contener mandatos que terminen el proceso actual. Una rutina externa siempre tiene que devolver el control al gestor de bases de datos DB2 sin terminar el proceso actual.
- Las bibliotecas, clases o ensamblajes de rutinas externas no deben actualizarse mientras la base de datos está activa, excepto en casos especiales. Si es necesario realizar una actualización mientras el gestor de bases de datos de DB2 está activo, y no es posible detener y volver a iniciar la instancia, cree la biblioteca, clase o ensamblaje nuevo para la rutina con uno diferente. A continuación, utilice la sentencia ALTER para cambiar el valor de la cláusula EXTERNAL NAME de la rutina externa de modo que haga referencia al nombre del archivo de biblioteca, clase o ensamblaje nuevo.
- La variable de entorno DB2CKPTR no está disponible en las rutinas externas. Las demás variables de entorno cuyos nombres empiezan por 'DB2' se capturan en el momento en que se inicia el gestor de bases de datos y pueden utilizarse en las rutinas externas.

- Algunas variables de entorno cuyos nombres no comienzan por 'DB2' no están disponibles para las rutinas externas delimitadas (fenced). Por ejemplo, la variable de entorno `LIBPATH` no puede utilizarse. Sin embargo, estas variables sí están disponibles para las rutinas externas que no están delimitadas (not fenced).
- Los valores de variables de entorno establecidos una vez iniciado el gestor de bases de datos de DB2 no están disponibles para las rutinas externas.
- Dentro de las rutinas externas, debe limitarse el uso de recursos protegidos, recursos a los que únicamente puede acceder un proceso a la vez. Si han de utilizarse, procure reducir la probabilidad de puntos muertos cuando dos rutinas externas intenten acceder al recurso protegido. Si se producen puntos muertos en dos o más rutinas, mientras se intenta acceder al recurso protegido, el gestor de bases de datos de DB2 no será capaz de detectar o resolver la situación. Esto ocasionará que los procesos de rutinas externas se queden colgados.
- La memoria para los parámetros de rutinas externas no debe asignarse explícitamente en el servidor de bases de datos DB2. El gestor de bases de datos de DB2 asigna automáticamente almacenamiento basándose en la declaración de parámetros de la sentencia CREATE para la rutina. No modifique ningún puntero del almacenamiento para los parámetros de las rutinas externas. Un intento de cambiar un puntero por un puntero del almacenamiento creado localmente puede tener como consecuencia fugas de memoria, corrupción de los datos o terminaciones anormales.
- No utilice datos estáticos ni globales en las rutinas externas. DB2 no puede garantizar que la memoria utilizada por las variables estáticas o globales no se toque entre las invocaciones de rutinas externas. Para las UDF y los métodos, puede emplear áreas reutilizables a fin de almacenar los valores que se utilizarán entre las invocaciones.
- Los valores de todos los parámetros de SQL se colocan en el almacenamiento intermedio. Esto significa que se realiza una copia del valor y se le pasa a la rutina externa. Si se efectúan cambios en los parámetros de entrada de una rutina externa, estos cambios no repercutirán en los valores de SQL ni en el proceso. Sin embargo, si una rutina externa graba, en un parámetro de entrada o de salida, más datos de los especificados en la sentencia CREATE, se habrá corrompido la memoria y es posible que la rutina termine anormalmente.
- El programa de utilidad LOAD no permite cargar en tablas que contienen columnas que hacen referencia a procedimientos delimitados. Si emite el mandato LOAD en una tabla de este tipo, recibirá el mensaje de error SQL1376N. Para evitar esta restricción, puede redefinir la rutina para que no esté protegida, o bien utilizar el programa de utilidad de importación.

Restricciones que se aplican solamente a los procedimientos externos

- Cuando se devuelvan conjuntos de resultados desde procedimientos almacenados anidados, podrá abrir un cursor con el mismo nombre en varios niveles de anidamiento. No obstante, las aplicaciones anteriores a la versión 8 sólo podrán acceder al primer conjunto de resultados que se haya abierto. Esta restricción no se aplica a los cursores abiertos con un nivel de paquete diferente.

Restricciones que se aplican solamente a las funciones externas

- Las funciones externas no pueden devolver conjuntos de resultados. Todos los cursores abiertos dentro de una función externa deben estar cerrados en el momento en que se complete la invocación a la llamada final de la función.
- Las asignaciones dinámicas de memoria en una rutina externa deben liberarse antes de que la rutina externa devuelva el control. De lo contrario, se producirán

fugas de memoria y el constante aumento del consumo de memoria de un proceso de DB2 puede tener como consecuencia que el sistema se quede sin memoria.

Para las funciones definidas por el usuario externas y los métodos externos, pueden utilizarse áreas reutilizables para asignar la memoria dinámica necesaria para la invocación de múltiples funciones. Si se utilizan áreas reutilizables de este modo, especifique el atributo FINAL CALL en la sentencia CREATE FUNCTION o CREATE METHOD. Así se garantiza que la memoria asignada se liberará antes de que la rutina devuelva el control.

Capítulo 6. Desarrollo de rutinas externas

Características de las funciones y métodos externos

Las funciones externas y los métodos externos proporcionan soporte para funciones que, en el caso de un determinado conjunto de datos de entrada, se podrían invocar múltiples veces y producir un conjunto de valores de salida.

Para obtener más información sobre las características de las funciones y métodos externos, consulte estos temas:

- "Funciones escalares externas"
- "Modelo de proceso de los métodos y las funciones escalares externas" en la página 141
- "Funciones de tabla externas" en la página 142
- "Modelo de proceso de las funciones de tabla externas" en la página 142
- "Modelo de ejecución de las funciones para Java" en la página 144
- "Áreas reutilizables para funciones externas y métodos" en la página 145
- "Áreas reutilizables en sistemas operativos de 32 bits y 64 bits" en la página 148

Estas características son exclusivas para las funciones y métodos externos y no atañen a las funciones de ni a los métodos de SQL.

Funciones escalares externas

Las funciones escalares externas tienen implementada su lógica en un lenguaje de programación externo.

Estas funciones pueden desarrollarse y utilizarse para ampliar el conjunto de funciones de SQL existentes y pueden invocarse del mismo modo que las funciones incorporadas de DB2, como por ejemplo, LENGTH y COUNT. Es decir, se puede hacer referencia a ellas en sentencias de SQL siempre que la expresión sea válida.

La ejecución de funciones escalares externas se lleva a cabo en el servidor de bases de datos DB2, aunque, a diferencia de las funciones escalares de SQL incorporadas o definidas por el usuario, la lógica de las funciones externas puede acceder al sistema de archivos del servidor de bases de datos, realizar llamadas al sistema o acceder a una red.

Las funciones escalares externas pueden leer datos de SQL, pero no pueden modificarlos.

Las funciones escalares externas se pueden invocar repetidamente para una sola referencia de la función y pueden mantener el estado entre estas invocaciones mediante un área reutilizable, que es un almacenamiento intermedio de memoria. Esto puede resultar potente si una función requiere una lógica de configuración inicial, pero costosa. La lógica de configuración se puede realizar en una primera invocación, utilizando el área reutilizable para almacenar algunos valores, cuyo acceso o actualización es posible en invocaciones siguientes de la función escalar.

Características de las funciones escalares externas

© Copyright IBM Corp. 1993, 2010

- Se puede hacer referencia a ellas como parte de una sentencia de SQL en cualquier parte en que esté soportada una expresión.
- La sentencia de SQL que realiza la invocación puede utilizar directamente la salida de una función escalar.
- Para las funciones escalares externas definidas por el usuario, se puede mantener el estado entre las invocaciones iterativas de la función empleando un área reutilizable.
- Pueden proporcionar una ventaja para el rendimiento cuando se utilizan en predicados, puesto que se ejecutan en el servidor. Si una función se puede aplicar a una fila candidata en el servidor, puede, a menudo, dejar de tomar en consideración la fila antes de transmitirla a la máquina cliente, con lo que se reduce la cantidad de datos que se deben pasar desde el servidor al cliente.

Limitaciones

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

Usos frecuentes

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

Lenguajes soportados

- C
- C++
- Java
- OLE
- Lenguajes CLR (Commmon Language Runtime) .NET

Nota:

1. Existe una capacidad limitada para crear funciones de agregación. Conocidas también como funciones de columna, estas funciones reciben un conjunto de valores semejantes (una columna de datos) y devuelven una sola respuesta. Una función de agregación definida por el usuario sólo se puede crear si su fuente es una función de agregación incorporada. Por ejemplo, si existe un tipo diferenciado `SHOESIZE` que está definido con el tipo base INTEGER, es posible definir una función, `AVG(SHOESIZE)`, como función de agregación basada en la función de agregación incorporada existente `AVG(INTEGER)`.
2. También puede crear funciones que devuelvan una fila. Éstas se conocen como funciones de fila y sólo se pueden utilizar como funciones de transformación para los tipos estructurados. La salida de una función de salida es una única fila.

Modelo de proceso de los métodos y las funciones escalares externas

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

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

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

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

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

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

Funciones de tabla externas

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

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

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

Modelo de proceso de las funciones de tabla externas

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

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

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

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

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

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

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

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

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

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

Para las UDF de tabla con NO FINAL CALL, la secuencia de llamadas OPEN, FETCH, FETCH, ..., CLOSE se repite para cada fila de table_1. Observe que cada llamada OPEN obtendrá un área reutilizable limpia. Puesto que la función de tabla no sabe, al final de cada exploración, si se producirán más exploraciones, la tiene que limpiar completamente durante el proceso de cierre (CLOSE). Esto puede resultar ineficaz si existe un proceso significativo de apertura de una sola vez que se debe repetir.

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

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

Modelo de ejecución de las funciones para Java

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

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

Nota:

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

Áreas reutilizables para funciones externas y métodos

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

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

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

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

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

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

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

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

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

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

   - Sólo la primera vez.

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

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

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

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

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

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

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

Así, para el ejemplo de *counter*, el último valor devuelto se puede conservar en el área reutilizable. Y, en el ejemplo de *match*, se puede conservar en el área reutilizable la lista de documentos, en caso de que el área reutilizable sea suficientemente grande; de lo contrario, se puede asignar memoria para la lista y conservar la dirección de la memoria adquirida en el área reutilizable. Las áreas reutilizables pueden tener una longitud variable: la longitud se define en la sentencia CREATE para la rutina.

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

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

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

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

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

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

Si especifica FINAL CALL, observe que la UDF o el método recibirán una llamada del tipo FIRST. Se puede utilizar esta posibilidad para adquirir e inicializar algún recurso persistente.

A continuación se muestra un ejemplo sencillo de Java de una UDF que utiliza un área reutilizable para calcular la suma de los cuadrados de las entradas de una columna. Este ejemplo toma una columna y devuelve una columna que contiene la suma acumulada de cuadrados desde el principio de la columna hasta la entrada de la fila actual:

```
CREATE FUNCTION SumOfSquares(INTEGER)
RETURNS INTEGER
EXTERNAL NAME 'UDFsrv!SumOfSquares'
DETERMINISTIC
NO EXTERNAL ACTION
FENCED
NOT NULL CALL
LANGUAGE JAVA
PARAMETER STYLE DB2GENERAL
NO SQL
SCRATCHPAD 10
FINAL CALL
DISALLOW PARALLEL
NO DBINFO@


// Suma de cuadrados utilizando la UDF Scratchpad
public void SumOfSquares(int inColumn,
                         int outSum)
```

```
throws Exception
{
  int sum = 0;
  byte[] scratchpad = getScratchpad();

  // variables a leer de área SCRATCHPAD
  ByteArrayInputStream byteArrayIn = new ByteArrayInputStream(scratchpad);
  DataInputStream dataIn = new DataInputStream(byteArrayIn);

  // variables a grabar en área SCRATCHPAD
  byte[] byteArrayCounter;
  int i;
  ByteArrayOutputStream byteArrayOut = new ByteArrayOutputStream(10);
  DataOutputStream dataOut = new DataOutputStream(byteArrayOut);

  switch(getCallType())
  {
    case SQLUDF_FIRST_CALL:
      // inicializar datos
   sum = (inColumn * inColumn);
      // guardar datos en área SCRATCHPAD
      dataOut.writeInt(sum);
      byteArrayCounter = byteArrayOut.toByteArray();
      for(i = 0; i < byteArrayCounter.length; i++)
      {
        scratchpad[i] = byteArrayCounter[i];
      }
      setScratchpad(scratchpad);
   break;
    case SQLUDF_NORMAL_CALL:
      // leer datos de área SCRATCHPAD
      sum = dataIn.readInt();
      // trabajar con datos
      sum = sum + (inColumn * inColumn);
      // guardar datos en área SCRATCHPAD
      dataOut.writeInt(sum);
      byteArrayCounter = byteArrayOut.toByteArray();
      for(i = 0; i < byteArrayCounter.length; i++)
      {
        scratchpad[i] = byteArrayCounter[i];
      }
      setScratchpad(scratchpad);
break;
  }
  // establecer valor de salida
  set(2, sum);
} // UDF SumOfSquares
```

Tenga en cuenta que se trata de una función incorporada de DB2 que realiza la misma función que la UDF SumOfSquares. Se ha elegido este ejemplo para demostrar el uso de un área reutilizable.

Áreas reutilizables en sistemas operativos de 32 bits y 64 bits

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

A continuación se muestra una declaración de estructura de ejemplo correspondiente a un área reutilizable:

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

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

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

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

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

```
struct spaddata
{
  sqlint32 int_var;
  sqlint64 bigint_var;
};
void SQL_API_FN routine( ..., struct sql_scratchpad* spad, ... )
{
  struct spaddata* scratchpad = (struct spaddata*)spad?data;
  /* Utilizar área reutilizable */
}
```

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

Soporte para el tipo de datos XML en las rutinas externas

Los procedimientos y las funciones externas escritas en los siguientes lenguajes de programación soportan parámetros y variables de tipo de datos XML:

- C
- C++
- COBOL
- Java
- Lenguajes .NET CLR

Las rutinas OLE y OLEDB externas no soportan parámetros de tipo de datos XML.

Los valores de tipo de datos XML se representan en el código de las rutinas externas de la misma manera que los tipos de datos CLOB.

Cuando se declaran parámetros de tipo de datos XML para las rutinas externas, las sentencias CREATE PROCEDURE y CREATE FUNCTION que se utilizarán para crear las rutinas en la base de datos deben especificar que el tipo de datos XML se

debe almacenar en forma de tipo de datos CLOB. El tamaño del valor de CLOB debe acercarse al tamaño del documento XML representado por el parámetro XML.

En la siguiente línea de código aparece una sentencia CREATE PROCEDURE de un procedimiento externo implementado en el lenguaje de programación C con un parámetro XML llamado parm1:

```
CREATE PROCEDURE myproc(IN parm1 XML AS CLOB(2M), IN parm2 VARCHAR(32000))
LANGUAGE C
FENCED
PARAMETER STYLE SQL
EXTERNAL NAME 'mylib!myproc';
```

Hay que tener en cuenta consideraciones parecidas al crear funciones definidas por usuario (UDF) externas, como se ve en el siguiente ejemplo:

```
CREATE FUNCTION myfunc (IN parm1 XML AS CLOB(2M))
RETURNS SMALLINT
LANGUAGE C
PARAMETER STYLE SQL
DETERMINISTIC
NOT FENCED
NULL CALL
NO SQL
NO EXTERNAL ACTION
EXTERNAL NAME 'mylib1!myfunc'
```

Los datos XML se materializan al pasarlos a procedimientos almacenados como parámetros IN, OUT o INOUT. Si utiliza procedimientos almacenados Java, es posible que se tenga que aumentar el tamaño de pila (parámetro de configuración JAVA_HEAP_SZ) de acuerdo con la cantidad y el tamaño de los argumentos XML, así como el número de procedimientos almacenados externos que se estén ejecutando simultáneamente.

En el código de las rutinas externas, los valores de los parámetros y variables XML se acceden, establecen y modifican de la misma manera que en las aplicaciones de bases de datos.

Escritura de rutinas

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

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

Antes de escribir una rutina, debe decidir lo siguiente:

- El tipo de rutina que necesita.
- El lenguaje de programación que utilizará para escribirla.
- Qué interfaz debe utilizar si necesita sentencias de SQL en la rutina.

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

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

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

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

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

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

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

Creación de rutinas externas

Las rutinas externas, incluidos procedimientos y funciones, se crean de forma parecida a las rutinas con otras implementaciones, aunque algunos pasos adicionales necesarios porque la implementación de la rutina requiere la codificación, compilación y despliegue del código fuente.

Elegirá implementar una rutina externa si:

- Desea encapsular lógica compleja en una rutina que acceda a la base de datos o que realice una acción fuera de la base de datos.
- Necesita que la lógica encapsulada se invoque desde cualquiera de estos elementos: diversas aplicaciones, el CLP, otra rutina (procedimiento, función (UDF) o método) o un activador.
- Se siente más cómodo al codificar esta lógica en un lenguaje de programación en lugar de utilizar sentencias de SQL y SQL PL.
- Necesita la lógica de rutina para realizar operaciones externas en la base de datos, tales como escribir o leer un archivo del servidor de bases de datos, la ejecución de otra aplicación, o lógica que no puede representarse con sentencias de SQL y SQL PL.

Requisitos previos

- Conocimiento de la implementación de rutinas externas. Para obtener información sobre las rutinas externas en general, consulte el tema:
  - "Rutinas externas" en la página 119
  - "Creación de rutinas externas" en la página 120
- Debe instalarse el Cliente DB2.
- El servidor de bases de datos debe ejecutar un sistema operativo que dé soporte a los compiladores del lenguaje de programación de la implementación elegida y al software de desarrollo.
- Los compiladores necesarios y el soporte de ejecución para el lenguaje de programación elegido deben estar instalados en el servidor de bases de datos
- Autorización para ejecutar la sentencia CREATE PROCEDURE, CREATE FUNCTION o CREATE METHOD.

Para obtener una lista de las restricciones asociadas con rutinas externas, consulte:

- "Restricciones para rutinas externas" en la página 135

Procedimiento

1. Codifique la lógica de la rutina en el lenguaje de programación elegido.
   - Para obtener información general sobre rutinas externas, características de las rutinas y la implementación de características de las rutinas, consulte los temas a los que se hace referencia en la sección Requisitos previos.
   - Utilice o importe algunos de los archivos de cabecera necesarios para dar soporte a la ejecución de sentencias de SQL.
   - Declare las variables y los parámetros correctamente utilizando tipos de datos del lenguaje de programación que se correlacionen con tipos de datos de SQL de DB2.
2. Los parámetros deben declararse de acuerdo con el formato requerido por el estilo de parámetro para el lenguaje de programación elegido. Si desea más información sobre los parámetros y las declaraciones de prototipo, consulte:
   - "Estilos de parámetros de rutinas externas" en la página 133
3. Construya el código en una biblioteca o archivo de clase.
4. Copie el archivo de clase o la biblioteca en el directorio *function* de DB2 en el servidor de bases de datos. Se recomienda almacenar los ensamblajes o las bibliotecas asociadas con las rutinas de DB2 en el directorio de función (function). Para conocer más acerca del directorio de función, consulte la cláusula EXTERNAL de una de las sentencias siguientes: CREATE PROCEDURE o CREATE FUNCTION.

Puede copiar el ensamblaje en otro directorio del servidor si lo desea, pero, para invocar satisfactoriamente la rutina, debe anotar el nombre de vía de acceso completamente calificado del ensamblaje porque lo necesitará en el paso siguiente.

5. Ejecute de forma dinámica o estática la sentencia CREATE de lenguaje SQL correspondiente para el tipo de rutina: CREATE PROCEDURE o CREATE FUNCTION.
   - Especifique la cláusula `LANGUAGE` con el valor adecuado correspondiente a la API o lenguaje de programación elegido. Ejemplos: CLR, C, JAVA.
   - Especifique la cláusula `PARAMETER STYLE` con el nombre del estilo de parámetro soportado que se ha implementado en el código de la rutina.
   - Especifique la cláusula `EXTERNAL` con el nombre del archivo de biblioteca, clase o ensamblaje que se ha de asociar a la rutina utilizando uno de los valores siguientes:
     – el nombre de vía de acceso calificado al completo del archivo de biblioteca, clase o ensamblaje de la rutina.
     – el nombre de vía de acceso relativa del archivo de biblioteca, clase o ensamblaje de la rutina con relación al directorio de función.

     Por omisión, DB2 buscará el archivo de biblioteca, clase o ensamblaje por el nombre en el directorio de función, a menos que se especifique un nombre de vía de acceso completamente calificado o relativo para el archivo en la cláusula EXTERNAL.
   - Especifique DYNAMIC RESULT SETS con un valor numérico si la rutina es un procedimiento y ha de devolver uno o varios conjuntos de resultados al llamador.
   - Especifique las demás cláusulas necesarias para caracterizar la rutina.

Para invocar la rutina externa, consulte Invocación de la rutina

Rutinas CLR (Common Language Runtime) .NET

En DB2, una rutina CLR (Common Language Runtime) es una rutina externa creada ejecutando una sentencia CREATE PROCEDURE o CREATE FUNCTION que hace referencia a un ensamblaje .NET como su cuerpo de código externo.

Los términos siguientes son importantes en el contexto de las rutinas CLR:

.NET Framework
Un entorno de desarrollo de aplicaciones de Microsoft que comprende tanto la CLR como la biblioteca de clases de .NET Framework diseñada para proporcionar un entorno de programación coherente con miras al desarrollo e integración de partes de código.

Common language runtime (CLR)
El intérprete de ejecución para todas las aplicaciones .NET Framework.

lenguaje intermedio (IL)
Tipo de código de bytes compilado que se interpreta mediante la CLR de .NET Framework. El código fuente de todos los lenguajes compatibles con .NET se compila en el código de bytes IL.

ensamblaje
Un archivo que contiene código de bytes IL. Puede ser una biblioteca o un ejecutable.

Es posible implementar rutinas CLR en los lenguajes que se puedan compilar en un ensamblaje IL. Estos lenguajes incluyen los siguientes, pero no están limitados a ellos: Managed C++, C#, Visual Basic y J#.

Antes de desarrollar una rutina CLR, es importante comprender los conceptos básicos de las rutinas y las características exclusivas y específicas de las rutinas CLR. Para informarse más acerca de las rutinas y de las rutinas CLR, consulte:

- "Beneficios del uso de rutinas" en la página 2
- "Representación de tipo de datos de SQL en rutinas CLR de .NET" en la página 155
- "Parámetros de rutinas .NET CLR" en la página 158
- "Devolución de conjuntos de resultados desde procedimientos .NET CLR" en la página 160
- "Restricciones de las rutinas CLR .NET" en la página 162
- "Errores relacionados con rutinas CLR .NET" en la página 173

Desarrollar una rutina CLR es fácil. Para obtener instrucciones paso a paso sobre cómo desarrollar una rutina CLR y ejemplos completos, consulte:

- "Creación de rutinas CLR .NET desde la ventana de mandatos de DB2" en la página 165
- "Ejemplos de procedimientos CLR de .NET en C#" en la página 176
- "Ejemplos de funciones CLR de .NET en C#" en la página 210

Soporte para el desarrollo de rutinas externas en lenguajes .NET CLR

Para desarrollar rutinas externas en lenguajes .NET CLR, y ejecutarlas correctamente, deberá utilizar sistemas operativos soportados, versiones de clientes y servidores de bases de datos de DB2, y software de desarrollo.

Las rutinas externas .NET CLR se pueden implementar en cualquier lenguaje que Microsoft .NET Framework pueda compilar en un conjunto IL. Estos lenguajes incluyen los siguientes, pero no están limitados a ellos: Managed C++, C#, Visual Basic y J#.

Puede desarrollar rutinas .NET CLR en los sistemas operativos siguientes:

- Windows 2000
- Windows XP (edición de 32 bits; edición de 64 bits comenzando con el Fix Pack 2)
- Windows Server 2003 (edición de 32 bits; edición de 64 bits comenzando con el Fix Pack 2)
- Windows Server 2008 (edición de 32 bits y edición de 64 bits comenzando con el Fix Pack 2)

Se debe instalar un cliente de servidor de datos de la versión 9 o posterior para el desarrollo de rutinas .NET CLR. El servidor de bases de datos debe estar ejecutando productos de base de datos DB2 Versión 9 o posterior.

Una versión soportada del software Microsoft .NET Framework debe también estar instalada en el mismo sistema que el servidor de bases de datos DB2. Microsoft .NET Framework se puede adquirir por separado o como parte de Microsoft .NET Framework Software Development Kit.

Herramientas para desarrollar rutinas .NET CLR

las herramientas pueden realizar la tarea de desarrollar rutinas .NET CLR que puedan interactuar con la base de datos de DB2 de forma más rápida y sencilla.

Las rutinas .NET CLR se pueden desarrollar en Microsoft Visual Studio .NET mediante herramientas gráficas disponibles en:

- IBM Database Add-Ins para Microsoft Visual Studio

Las siguientes interfaces de líneas de mandatos proporcionadas con DB2, están también disponibles para desarrollar rutinas .NET CLR en DB2:

- Procesador de línea de mandatos de DB2 (DB2 CLP)
- Ventana de mandatos de DB2

Diseño de rutinas de CLR .NET

Al diseñar rutinas .NET CLR, deberá tener en cuenta consideraciones generales sobre el diseño de rutinas externas y consideraciones sobre el diseño específico de .NET CLR.

Conocimientos y experiencia en el desarrollo de aplicaciones de .NET y un conocimiento general de las rutinas externas. Los temas siguientes proporcionan parte de la información necesaria sobre requisitos previos.

Para obtener más información sobre las funciones y usos de las rutinas externas consulte:

- Rutinas externas

Para obtener más información sobre las características de las rutinas .NET CLR, consulte:

- rutinas .NET CLR

Con el conocimiento de los requisitos previos, el diseño de rutinas de SQL incorporado consiste principalmente en conocer las funciones y características exclusivas de las rutinas .NET CLR:

- Conjuntos de inclusión que proporcionan soporte para le ejecución de sentencias de SQL en rutinas .NET CLR (IBM.Data.DB2)
- Tipos de datos de SQL soportados en rutinas .NET CLR
- Parámetros de rutinas .NET CLR
- Devolución de conjuntos de resultados desde rutinas .NET CLR
- Valores de modalidad de control de ejecución y seguridad para rutinas .NET CLR
- Restricciones de rutinas .NET CLR
- Devolución de conjuntos de resultados desde procedimientos .NET CLR

Después de conocer las características de .NET CLR, puede: "Crear rutinas .NET CLR".

Representación de tipo de datos de SQL en rutinas CLR de .NET

Las rutinas CLR de .NET pueden hacer referencia a valores del tipo de datos de SQL como parámetros de rutina, valores de parámetros que deben utilizarse como parte de la ejecución de la sentencia de SQL y como variables. No obstante, los valores del tipo de datos de SQL de IBM apropiados, los valores de tipo de datos para IBM Data Server Provider para .NET y los valores de tipo de datos de .NET

Framework deben utilizarse para garantizar que no se produzca ningún truncamiento o pérdida de datos al acceder o recuperar los valores.

Para especificaciones de parámetros de rutina dentro las sentencias CREATE PROCEDURE o CREATE FUNCTION utilizadas para crear rutinas CLR .NET, se utilizan valores del tipo de datos de SQL de DB2. La mayoría de tipos de datos de SQL se pueden especificar para parámetros de rutina, no obstante, existen algunas excepciones.

Para especificar valores de parámetros que deban utilizarse como parte de una sentencia de SQL que deba ejecutarse, debe utilizar objetos de IBM Data Server Provider para .NET. El objeto `DB2Parameter` se utiliza para representar un parámetro que deba añadirse a un objeto `DB2Command` que represente una sentencia de SQL. Al especificar el valor de tipo de datos para el parámetro, deben utilizarse los valores del tipo de datos de IBM Data Server Provider para .NET disponibles en el espacio de nombres `IBM.Data.DB2Types`. El espacio de nombres `IBM.Data.DB2Types` proporciona clases y estructuras para representar cada tipo de datos SQL de DB2 soportado.

Para parámetros y variables locales que es posible que temporalmente mantengan valores de tipos de datos de SQL, debe utilizar los tipos de datos de IBM Data Server Provider para .NET, tal y como se ha definido en el espacio de nombres IBM.Data.DB2Types.

Nota: La estructura `dbinfo` pasa a las funciones y procedimientos CLR como un parámetro. El área reutilizable y el tipo de llamada para las UDF CLR también pasan a las rutinas CLR como parámetros. Si desea información sobre los tipos de datos CLR adecuados para estos parámetros, consulte el tema relacionado:

- Parámetros de las rutinas CLR

La siguiente tabla muestra correlaciones entre tipos de datos de DB2Type, tipos de datos de DB2, tipos de datos de Informix, Microsoft .NET Framework y clases y estructuras de DB2Types.

Categoría	Clases y estructuras de DB2Types	Tipo de datos DB2Type	Tipo de datos DB2	Tipo de datos Informix	Tipo de datos .NET
Numeric	DB2Int16	SmallInt	SMALLINT	BOOLEAN, SMALLINT	Int16
	DB2Int32	Integer	INT	INTEGER, INT, SERIAL	Int32
	DB2Int64	BigInt	BIGINT	INT8, SERIAL8	Int64
	DB2Real, DB2Real370	Real	REAL	REAL, SMALLFLOAT	Single
	DB2Double	Double	DOUBLE PRECISION	DECIMAL (=31), DOUBLE PRECISION	Double
	DB2Double	Float	FLOAT	DECIMAL (32), FLOAT	Double
	DB2Decimal	Decimal	DECIMAL	MONEY	Decimal
	DB2DecimalFloat	DecimalFloat	DECFLOAT (16|34)[14]		Decimal
	DB2Decimal	Numeric	DECIMAL	DECIMAL (=31), NUMERIC	Decimal
Date/Time	DB2Date	Date	DATE	DATETIME (date precision)	Datetime
	DB2Time	Time	TIME	DATETIME (time precision)	TimeSpan
	DB2TimeStamp	Timestamp	TIMESTAMP	DATETIME (time and date precision)	DateTime
XML	DB2Xml	Xml[2]	XML		Byte[]
Character data	DB2String	Char	CHAR	CHAR	String
	DB2String	VarChar	VARCHAR	VARCHAR	String
	DB2String	LongVarChar[1]	LONG VARCHAR	LVARCHAR	String
Datos binarios	DB2Binary	Binary	CHAR FOR BIT DATA		Byte[]
	DB2Binary	Binary[3]	BINARY		Byte[]
	DB2Binary	VarBinary[3]	VARBINARY		Byte[]
	DB2Binary	LongVarBinary[1]	LONG VARCHAR FOR BIT DATA		Byte[]
Graphic data	DB2String	Graphic	GRAPHIC		String
	DB2String	VarGraphic	VARGRAPHIC		String
	DB2String	LongVarGraphic[1]	LONG VARGRAPHIC		String
Datos LOB	DB2Clob	Clob	CLOB	CLOB, TEXT	String
	DB2Blob	Blob	BLOB	BLOB, BYTE	Byte[]
	DB2Clob	DbClob	DBCLOB		String
Row ID	DB2RowId	RowId	ROWID		Byte[]

1. Estos tipos de datos no están soportados como parámetros en rutinas de tiempo de ejecución de lenguaje común de DB2 .NET.
2. Una propiedad DB2ParameterClass.ParameterName del tipo DB2Type.Xml puede aceptar variables de los siguientes tipos: String, byte[], DB2Xml y XmlReader.
3. Estos tipos de datos sólo se aplican a DB2 UDB para z/OS.
4. Este tipo de datos sólo se soporta para DB2 para z/OS Versión 9 y releases posteriores y para DB2 para Linux, UNIX y Windows Versión 9.5 y releases posteriores.

Parámetros de rutinas .NET CLR

La declaración de parámetros de las rutinas .NET CLR debe cumplir con los requisitos de uno de los estilos de parámetros soportados y debe respetar los requisitos de palabra clave de parámetro del lenguaje .NET determinado que se utilice para la rutina. Si la rutina ha de emplear un área reutilizable o la estructura `dbinfo` o ha de tener la interfaz de parámetros PROGRAM TYPE MAIN, existen detalles adicionales a considerar. Este tema aborda todas las consideraciones sobre los parámetros CLR.

Estilos de parámetros soportados para las rutinas CLR

El estilo de parámetro de la rutina se debe especificar durante la creación de la rutina en la cláusula EXTERNAL de la sentencia CREATE para la rutina. El estilo de parámetro se debe reflejar adecuadamente en la implementación del código de la rutina CLR externa. Están soportados los estilos de parámetros siguientes de DB2 para las rutinas CLR:

- `SQL` (Soportado para procedimientos y funciones)
- `GENERAL` (Soportado para procedimientos únicamente)
- `GENERAL WITH NULLS` (Soportado para procedimientos únicamente)
- `DB2SQL` (Soportado para procedimientos y funciones)

Si desea más información sobre estos estilos de parámetros, consulte el tema:

- Estilos de parámetros para rutinas externas

Indicadores de nulo de los parámetros de rutinas CLR

Si el estilo de parámetro elegido para una rutina CLR requiere que se especifiquen indicadores de nulo para los parámetros, los indicadores de nulo se han de pasar a la rutina CLR como valores de tipo System.Int16 o en un valor System.Int16[] cuando el estilo de parámetro exija un vector de indicadores de nulo.

Cuando el estilo de parámetro impone que se pasen los indicadores de nulo a la rutina como parámetros diferenciados, tal como requiere el estilo de parámetro SQL, se necesita un indicador de nulo System.Int16 para cada parámetro.

En los lenguajes .NET, los parámetros diferenciados deben ir precedidos de una palabra clave para indicar si el parámetro se pasa por valor o por referencia. La misma palabra clave que se utilice para un parámetro de rutina se debe utilizar para el parámetro de indicador de nulo asociado. Las palabras clave que indican si un argumento se pasa por valor o por referencia se describen con más detalle a continuación.

Si desea obtener más información sobre el estilo de parámetro SQL y los otros estilos de parámetros soportados, consulte el tema:

- Estilos de parámetros para rutinas externas

Pase de parámetros de rutinas CLR por valor o por referencia

Las rutinas de lenguaje .NET que se compilan en el código de bytes del lenguaje intermedio (IL) requieren que los parámetros vayan precedidos de palabras clave que indiquen las propiedades determinadas del parámetro, tales como si el parámetro se pasa por valor o por referencia, o si es un parámetro de sólo entrada o de sólo salida.

Las palabras clave de parámetro son específicas del lenguaje .NET. Por ejemplo, para pasar un parámetro por referencia en C#, la palabra clave de parámetro es ref, mientras que, en Visual Basic, un parámetro por referencia se indica mediante la palabra clave byRef. Las palabras clave se deben emplear para indicar el uso del parámetro de SQL (IN, OUT, INOUT) que se ha especificado en la sentencia CREATE para la rutina.

Se imponen las normas siguientes al aplicar palabras clave de parámetro a los parámetros de rutinas de lenguaje .NET en las rutinas de DB2:

- Los parámetros de tipo IN se deben declarar *sin* palabra clave de parámetro en C#, y se deben declarar con la palabra clave byVal en Visual Basic.
- Los parámetros de tipo INOUT se deben declarar con la palabra clave específica del lenguaje que indique que el parámetro se pasa por referencia. En C#, la palabra clave adecuada es ref. En Visual Basic, la palabra clave adecuada es byRef.
- Los parámetros de tipo OUT se deben declarar con la palabra clave específica del lenguaje que indique que el parámetro es de sólo salida. En C#, utilice la palabra clave out. En Visual Basic, el parámetro se debe declarar con la palabra clave byRef. Los parámetros de sólo salida siempre deben tener asignado un valor antes de que la rutina vuelva al llamador. Si la rutina no asigna un valor al parámetro de sólo salida, se generará un error cuando se compile la rutina .NET.

A continuación, se muestra el aspecto de un prototipo de procedimiento en C# del estilo de parámetro SQL para una rutina que devuelve un solo parámetro de salida language.

```
public static void Counter  (out String language,
                             out Int16  languageNullInd,
                             ref String sqlState,
                                 String funcName,
                                 String funcSpecName,
                             ref String sqlMsgString,
                                 Byte[] scratchPad,
                                 Int32  callType);
```

Es evidente que se implementa el estilo de parámetro SQL por el parámetro de indicador de nulo adicional, languageNullInd, asociado con el parámetro de salida language, los parámetros para pasar el SQLSTATE, el nombre de rutina, el nombre de rutina específico y un mensaje de error de SQL opcional definido por el usuario. Se han especificado palabras clave para los parámetros del modo siguiente:

- En C#, no es necesaria ninguna palabra clave de parámetro para los parámetros de sólo entrada.
- En C#, la palabra clave 'out' indica que la variable es un parámetro de sólo salida y que el llamador no ha inicializado su valor.
- En C#, la palabra clave 'ref' indica que el llamador ha inicializado el parámetro y que la rutina puede modificar este valor opcionalmente.

Consulte la documentación específica del lenguaje .NET en relación con el pase de parámetros para informarse sobre las palabras clave de parámetro en ese lenguaje.

Nota: DB2 controla la asignación de memoria para todos los parámetros y mantiene las referencias CLR a todos los parámetros pasados a o desde una rutina.

No es necesario ningún marcador de parámetro para los conjuntos de resultados de procedimiento

Los marcadores de parámetros no son necesarios en la declaración de un procedimiento para un conjunto de resultados que se va a devolver al llamador. Cualquier sentencia de cursor que no se haya cerrado desde dentro de un procedimiento almacenado CLR se devolverá a su llamador como conjunto de resultados.

Si desea más información sobre los conjuntos de resultados en las rutinas CLR, consulte:

•

Estructura Dbinfo como parámetro CLR

La estructura `dbinfo` utilizada para pasar parámetros adicionales de información de base de datos a y desde una rutina está soportada para las rutinas CLR mediante el uso de una clase `dbinfo` de IL. Esta clase contiene todos los elementos que se encuentran en la estructura `sqludf_dbinfo` del lenguaje `C`, a excepción de los campos de longitud asociados con las series. La longitud de cada serie se puede encontrar utilizando la propiedad de lenguaje .NET `Length` de la serie determinada.

Para acceder a la clase `dbinfo`, simplemente incluya el conjunto `IBM.Data.DB2` en el archivo que contenga la rutina y añada un parámetro del tipo `sqludf_dbinfo` a la signatura de la rutina, en la posición especificada por el estilo de parámetro utilizado.

Área reutilizable de UDF como parámetro CLR

Si se solicita un área reutilizable para una función definida por el usuario, se pasa a la rutina como parámetro `System.Byte[]` del tamaño especificado.

Parámetro de llamada final o de tipo de llamada de UDF CLR

Para las funciones definidas por el usuario que han solicitado un parámetro de llamada final o para las funciones de tabla, el parámetro de tipo de llamada se pasa a la rutina como tipo de datos `System.Int32`.

PROGRAM TYPE MAIN está soportado para los procedimientos CLR

PROGRAM TYPE MAIN está soportado para los procedimientos .NET CLR. Los procedimientos definidos para el uso de Program Type MAIN deben tener la signatura siguiente:

```
void functionname(Int32 NumParams, Object[] Params)
```

Devolución de conjuntos de resultados desde procedimientos .NET CLR

Es posible desarrollar procedimientos CLR que devuelvan conjuntos de resultados a la rutina o aplicación que realiza la llamada. No se pueden devolver conjuntos de resultados desde las funciones de CLR (UDF).

La representación en .NET de un conjunto de resultados es un objeto `DB2DataReader` que se puede devolver desde una de las diversas llamadas de ejecución de un objeto `DB2Command`. Se puede devolver cualquier objeto `DB2DataReader` cuyo método `Close()` no se haya llamado explícitamente antes de la

devolución del procedimiento. El orden en que se devuelven los conjuntos de resultados al llamador es el mismo orden en que se han creado las instancias de los objetos DB2DataReader. No se requieren parámetros adicionales en la definición de función para devolver un conjunto de resultados.

Un conocimiento de cómo se crean las rutinas CLR le ayudará a seguir los pasos del procedimiento siguiente sobre la devolución de resultados desde un procedimiento CLR.

- "Creación de rutinas CLR .NET desde la ventana de mandatos de DB2" en la página 165

Para devolver un conjunto de resultados de un procedimiento CLR:

1. En la sentencia CREATE PROCEDURE para la rutina CLR, debe especificar, junto con cualquier otra cláusula apropiada, la cláusula DYNAMIC RESULT SETS con un valor que equivalga al número de conjuntos de resultados que debe devolver el procedimiento.
2. Los marcadores de parámetros no son necesarios en la declaración de un procedimiento para un conjunto de resultados que se va a devolver al llamador.
3. En la implementación en el lenguaje .NET de la rutina CLR, cree un objeto DB2Connection, un objeto DB2Command y un objeto DB2Transaction. El objeto DB2Transaction es el encargado de retrotraer y confirmar las transacciones de base de datos.
4. Inicialice la propiedad Transaction del objeto DB2Command para el objeto DB2Transaction.
5. Asigne una consulta de serie a la propiedad CommandText del objeto DB2Command que defina el conjunto de resultados que desea devolver.
6. Cree una instancia de un DB2DataReader y asígnele el resultado de la invocación del método ExecuteReader del objeto DB2Command. El conjunto de resultados de la consulta estará incluido en el objeto DB2DataReader.
7. No ejecute el método Close() del objeto DB2DataReader en ningún punto anterior a la devolución del procedimiento al llamador. El objeto DB2DataReader se devolverá todavía abierto como un conjunto de resultados al llamador.

   Cuando se deja abierto más de un DB2DataReader después de la devolución de un procedimiento, los DB2DataReader se devuelven al llamador siguiendo el orden de su creación. Sólo se devolverá al llamador el número de conjuntos de resultados especificado en la sentencia CREATE PROCEDURE.
8. Compile el procedimiento de lenguaje .NET CLR e instale el conjunto en la ubicación especificada por la cláusula EXTERNAL de la sentencia CREATE PROCEDURE. Ejecute la sentencia CREATE PROCEDURE para el procedimiento CLR, si todavía no lo ha hecho.
9. Una vez que haya instalado el conjunto del procedimiento CLR en la ubicación adecuada y haya ejecutado la sentencia CREATE PROCEDURE satisfactoriamente, puede invocar el procedimiento con la sentencia CALL para ver la devolución de conjuntos de resultados al llamador.

Modalidades de seguridad y de ejecución para rutinas CLR

Como administrador de bases de datos o desarrollador de aplicaciones, es posible que desee proteger los activos asociados a las rutinas externas de DB2 frente a manipulaciones no deseadas y restringir las acciones de las rutinas en el momento de la ejecución. Las rutinas CLR (Common Language Runtime) .NET de DB2 dan soporte a la especificación de una modalidad de control de ejecución que identifica los tipos de acciones que una rutina puede realizar en el momento de la ejecución. En el momento de la ejecución, DB2 puede detectar si la rutina intenta realizar

acciones que quedan fuera del ámbito de su modalidad de control de ejecución especificada, lo que puede resultar de ayuda para determinar si se ha puesto en peligro un activo.

Para establecer la modalidad de control de ejecución de una rutina CLR, especifique la cláusula opcional EXECUTION CONTROL en la sentencia CREATE correspondiente a la rutina. Las modalidades válidas son:

- SAFE
- FILEREAD
- FILEWRITE
- NETWORK
- UNSAFE

Para modificar la modalidad de control de ejecución de una rutina CLR existente, ejecute la sentencia ALTER PROCEDURE o ALTER FUNCTION.

Si no se especifica la cláusula EXECUTION CONTROL para una rutina CLR, por omisión la rutina CLR se ejecuta utilizando la modalidad de control de ejecución más restrictiva: SAFE. Las rutinas que se crean con esta modalidad de control de ejecución sólo pueden acceder a los recursos controlados por el gestor de bases de datos. Las modalidades de control de ejecución menos restrictivas permiten a una rutina acceder a archivos (FILEREAD o FILEWRITE) o realizar operaciones de red tales como acceder a una página web (NETWORK). La modalidad de control de ejecución UNSAFE especifica que no se debe colocar ninguna restricción en el comportamiento de la rutina. Las rutinas definidas con la modalidad de control de ejecución UNSAFE pueden ejecutar código binario.

Estas modalidades representan una jerarquía de acciones permitidas y una modalidad de nivel superior incluye las acciones permitidas por debajo de la misma en la jerarquía. Por ejemplo, la modalidad de control de ejecución NETWORK permite a una rutina acceder a páginas Web en Internet, a archivos de lectura y grabación y a recursos de acceso controlados por el gestor de bases de datos. Se recomienda utilizar la modalidad de control de ejecución más restrictiva posible y evitar utilizar la modalidad UNSAFE.

Si DB2 detecta en el momento de la ejecución que una rutina CLR está intentando una acción que queda fuera del ámbito de su modalidad de control de ejecución, DB2 devuelve un error (SQLSTATE 38501).

La cláusula EXECUTION CONTROL sólo se puede especificar para rutinas CLR LANGUAGE. El ámbito de aplicación de la cláusula EXECUTION CONTROL se limita a la propia rutina CLR .NET y no se aplica a ninguna otra rutina a la que pueda llamar.

Consulte la sintaxis de la sentencia CREATE correspondiente al tipo de rutina adecuado para ver una descripción completa de las modalidades de control de ejecución soportadas.

Restricciones de las rutinas CLR .NET

Las restricciones generales de implementación que se aplican a todas las rutinas externas o a determinadas clases de rutinas (procedimiento o UDF) también se aplican a las rutinas CLR. Existen algunas restricciones que son particulares de las rutinas CLR. Estas restricciones se listan aquí.

La sentencia CREATE METHOD con la cláusula LANGUAGE CLR no está soportada.

No se pueden crear métodos externos para tipos estructurados de DB2 que hacen referencia a un conjunto CLR. El uso de una sentencia CREATE METHOD que especifique la cláusula LANGUAGE con el valor CLR no está soportado.

Los procedimientos CLR no se pueden implementar como procedimientos NOT FENCED

Los procedimientos CLR no se pueden ejecutar como procedimientos no protegidos. La sentencia CREATE PROCEDURE para un procedimiento CLR no puede especificar la cláusula NOT FENCED.

La cláusula EXECUTION CONTROL restringe la lógica incluida en la rutina

La cláusula EXECUTION CONTROL y el valor asociado determinan los tipos de lógica y las operaciones que pueden ejecutarse en una rutina .NET CLR. Por omisión el valor de la cláusula EXECUTION CONTROL se establece en SAFE. Para la lógica de rutina que lee archivos, graba en archivos o que accede a Internet, debe especificarse un valor diferente al valor por omisión para la cláusula EXECUTION CONTROL que sea menos restrictivo.

La precisión decimal máxima es 29, la escala decimal máxima es 28 en una rutina CLR

El tipo de datos DECIMAL en DB2 se representa con una precisión de 31 dígitos y una escala de 28 dígitos. El tipo de datos System.Decimal de CLR .NET está limitado a una precisión de 29 dígitos y a una escala de 28 dígitos. Por lo tanto, las rutinas CLR externas de DB2 no pueden asignar un valor a un tipo de datos System.Decimal que sea mayor que `(2^96)-1`, que es el valor más alto que puede representarse utilizando una precisión de 29 dígitos y una escala de 28 dígitos. DB2 devolverá un error de tiempo de ejecución (SQLSTATE 22003, SQLCODE -413) si se produce una asignación de este tipo. En el momento de la ejecución de la sentencia CREATE para la rutina, si un parámetro de tipo de datos DECIMAL está definido con una escala mayor que 28, DB2 devolverá un error (SQLSTATE 42613, SQLCODE -628).

Si es necesario que su rutina manipule valores decimales con la precisión y escala máximas soportadas por DB2, puede implementar la rutina externa en un lenguaje de programación distinto, como, por ejemplo, Java.

Algunos tipos de datos no se pueden utilizar en rutinas CLR

Los siguientes tipos de datos de SQL de DB2 no están soportados en las rutinas CLR:

- LONG VARCHAR
- LONG VARCHAR FOR BIT DATA
- LONG VARGRAPHIC
- ROWID

Las rutinas CLR de 32 bits no se pueden ejecutar en instancias de 64 bits

Las rutinas CLR de 32 bits no se pueden ejecutar en instancias de 64 bits. A partir de la Versión 9.5 Fix Pack 2 se pueden ejecutar rutinas CLR de 64 bits en instancias de 64 bits.

No se puede utilizar .NET CLR para implementar plugins de seguridad

No se puede utilizar CLR .NET para compilar y enlazar código fuente para bibliotecas de plugins de seguridad.

Creación de rutinas .NET CLR

La creación de rutinas Java incluye lo siguiente:

- Ejecutar una sentencia CREATE que define la rutina en un servidor de bases de datos de DB2
- Desarrollar la implementación de la rutina que se corresponde con la definición de la rutina.

A continuación se citan las formas en las que puede crear rutinas Java:

- Mediante las herramientas gráficas que se proporcionan con IBM Database Add-Ins para Microsoft Visual Studio
- Utilizar la ventana de mandatos de DB2

En general lo más fácil es crear rutinas .NET CLR utilizando IBM Database Add-Ins para Microsoft Visual Studio. Si no puede utilizarse esta opción, la ventana de mandatos de DB2 proporciona un soporte similar mediante una interfaz de línea de mandatos.

Requisitos previos

- Revise el "Rutinas CLR (Common Language Runtime) .NET" en la página 153.
- Asegúrese de que tiene acceso a un servidor de DB2 Versión 9, incluyendo instancias y bases de datos.
- Asegúrese de que el sistema operativo está en un nivel de versión soportado por los productos de base de datos DB2.
- Asegúrese de que el software de desarrollo Microsoft .NET esté en un nivel de versión que esté soportado para el desarrollo de las rutinas .NET CLR.
- Autorización para ejecutar la sentencia CREATE PROCEDURE o CREATE FUNCTION.

Para obtener una lista de las restricciones asociadas con rutinas CLR, consulte:

- "Restricciones de las rutinas CLR .NET" en la página 162

Cree rutinas .NET CLR desde una de las interfaces siguientes:

- Visual Studio .NET cuando IBM Database Add-Ins for Microsoft Visual Studio también están instalados. Cuando se haya instalado Add-In, las herramientas gráficas integradas en Visual Studio .NET estarán disponibles para crear rutinas .NET CLR que funcionan en servidores de bases de datos DB2.
- Ventana de mandatos de DB2

Si desea crear rutinas CLR .Net desde la ventana de mandatos de DB2, consulte:

- "Creación de rutinas CLR .NET desde la ventana de mandatos de DB2"

Creación de rutinas CLR .NET desde la ventana de mandatos de DB2

Los procedimientos y funciones que hacen referencia a un conjunto de lenguaje intermedio se crean de la misma forma que cualquier rutina externa. Elegirá implementar una rutina externa en un lenguaje .NET si:

- Desea encapsular lógica compleja en una rutina que acceda a la base de datos o que realice una acción fuera de la base de datos.
- Necesita que la lógica encapsulada se invoque desde cualquiera de estos elementos: diversas aplicaciones, el CLP, otra rutina (procedimiento, función (UDF) o método) o un activador.
- Se siente más cómodo al codificar esta lógica en un lenguaje .NET.

Requisitos previos

- Conocimiento de la implementación de rutinas CLR. Para informarse sobre las rutinas CLR en general y sobre las características CLR, consulte:
  - "Rutinas CLR (Common Language Runtime) .NET" en la página 153
- El servidor de bases de datos debe ejecutar un sistema operativo Windows que dé soporte a Microsoft .NET Framework.
- Es necesario que haya instalada una versión soportada del software Microsoft .NET Framework en el servidor. .NET Framework se puede adquirir por separado o como parte de Microsoft .NET Framework Software Development Kit.
- Se deben instalar las versiones siguientes de DB2:
  - Servidor: DB2 8.2 o un release posterior.
  - Cliente: Cualquier cliente que se pueda conectar a una instancia de DB2 8.2 será capaz de invocar una rutina CLR. Es recomendable instalar DB2 Versión 7.2 o un release posterior en el cliente.
- Autorización para ejecutar la sentencia CREATE correspondiente a la rutina externa. Si desea saber cuáles son los privilegios necesarios para ejecutar la sentencia CREATE PROCEDURE o CREATE FUNCTION, consulte los detalles relativos a la sentencia pertinente.

Para obtener una lista de las restricciones asociadas con rutinas CLR, consulte:

- "Restricciones de las rutinas CLR .NET" en la página 162

Procedimiento

1. Codifique la lógica de la rutina en cualquier lenguaje CLR soportado.
   - Para obtener información general sobre las rutinas CLR .NET y sus características, consulte los temas referidos en el apartado Requisitos previos.
   - Utilice o importe el conjunto `IBM.Data.DB2` si la rutina ha de ejecutar SQL.
   - Declare las variables del lenguaje principal y los parámetros correctamente utilizando tipos de datos que se correlacionen con tipos de datos de SQL de DB2. Para una correlación de tipo de datos entre tipos de datos DB2 y .NET:
     - "Representación de tipo de datos de SQL en rutinas CLR de .NET" en la página 155

- Los parámetros y los indicadores de nulo de parámetro se deben declarar utilizando uno de los estilos de parámetros soportados por DB2 y de acuerdo con los requisitos de los parámetros de las rutinas CLR .NET. Asimismo, las áreas reutilizables para las UDF y la clase DBINFO pasan a las rutinas CLR como parámetros. Si desea más información sobre los parámetros y las declaraciones de prototipo, consulte:
  - "Parámetros de rutinas .NET CLR" en la página 158
- Si la rutina es un procedimiento y desea devolver un conjunto de resultados al llamador de la rutina, no es necesario ningún parámetro para el conjunto de resultados. Si desea más información sobre la devolución de conjuntos de resultados desde rutinas CLR:
  - "Devolución de conjuntos de resultados desde procedimientos .NET CLR" en la página 160
- Establezca un valor de retorno de rutina si es necesario. Las funciones escalares CLR requieren que se establezca un valor de retorno antes de la devolución. Las funciones de tabla CLR requieren que se especifique un código de retorno como parámetro de salida para cada invocación de la función de tabla. Los procedimientos CLR no realizan una devolución con un valor de retorno.

2. Cree el código en un conjunto de lenguaje intermedio (IL) para que se ejecute mediante la CLR. Para obtener información sobre cómo construir rutinas CLR .NET que accedan a DB2, consulte el tema siguiente:
   - "Creación de código de rutinas .NET CLR" en la página 167
3. Copie el ensamblaje en el directorio *function* de DB2 en el servidor de bases de datos. Se recomienda almacenar los ensamblajes o las bibliotecas asociadas con las rutinas de DB2 en el directorio de función (function). Para conocer más acerca del directorio de función, consulte la cláusula EXTERNAL de una de las sentencias siguientes: CREATE PROCEDURE o CREATE FUNCTION.

   Puede copiar el conjunto en otro directorio del servidor si lo desea, pero, para invocar satisfactoriamente la rutina, debe anotar el nombre de vía de acceso completamente calificado del conjunto porque lo necesitará en el paso siguiente.
4. Ejecute de forma dinámica o estática la sentencia CREATE de lenguaje SQL correspondiente para el tipo de rutina: CREATE PROCEDURE o CREATE FUNCTION.
   - Especifique la cláusula `LANGUAGE` con el valor: `CLR`.
   - Especifique la cláusula `PARAMETER STYLE` con el nombre del estilo de parámetro soportado que se ha implementado en el código de la rutina.
   - Especifique la cláusula `EXTERNAL` con el nombre del conjunto que se ha de asociar con la rutina utilizando uno de los valores siguientes:
     - el nombre de vía de acceso completamente calificado del conjunto de rutinas.
     - el nombre de vía de acceso relativo del conjunto de rutinas en relación con el directorio de función.

     Por omisión, DB2 buscará el conjunto por el nombre en el directorio de función, a menos que se especifique un nombre de vía de acceso completamente calificado o relativo para la biblioteca en la cláusula EXTERNAL.

     Cuando se ejecute la sentencia CREATE, si DB2 no encuentra el conjunto especificado en la cláusula EXTERNAL, se recibirá un error (SQLCODE -20282) con el código de razón 1.

- Especifique la cláusula DYNAMIC RESULT SETS con un valor entero que equivalga al número máximo de conjuntos de resultados que debe devolver la rutina.
- No es posible especificar la cláusula NOT FENCED para los procedimientos CLR. Por omisión, los procedimientos CLR se ejecutan como procedimientos FENCED.

Creación de código de rutinas .NET CLR

Una vez se ha escrito el código de implementación de rutina .NET, se debe crear para que el conjunto de rutinas se pueda desplegar y la rutina se pueda invocar. Los pasos a seguir para crear rutinas .NET CLR son parecidos a los necesarios para crear cualquier rutina externa, aunque hay algunas diferencias.

Hay tres formas de crear rutinas .NET CLR:

- Mediante las herramientas gráficas que se proporcionan con IBM Database Add-Ins para Microsoft Visual Studio
- Mediante archivos de proceso por lotes de ejemplo de DB2
- Entrando mandatos desde una ventana de mandatos de DB2

Los scripts de creación de ejemplo de DB2 y los archivos de proceso por lotes correspondientes a rutinas están diseñados para crear rutinas de ejemplo de DB2 (procedimientos y funciones definidas por el usuario), así como rutinas creadas por el usuario para un determinado sistema operativo utilizando los compiladores soportados por omisión.
Hay un conjunto independiente de scripts de creación de ejemplo de DB2 y de archivos de proceso por lotes para C# y Visual Basic. En general, es más fácil crear rutinas de .NET CLR incorporado utilizando las herramientas gráficas o los scripts de creación, que se pueden modificar fácilmente si hace falta; sin embargo, suele resultar útil saber también cómo crear rutinas a partir de la ventana de mandatos de DB2.

Creación de código de rutinas CLR (Common Languaje Runtime) de .NET mediante scripts de creación de ejemplo

La creación de código fuente para rutinas (Common Language Runtime) de .NET es una subtarea de creación de rutinas .NET CLR. Esta tarea se puede realizar de forma rápida y fácil mediante los archivos por lotes de ejemplo de DB2. Los scripts de creación de ejemplo se pueden utilizar para el código fuente con sentencias de SQL o sin ellas. Los scripts de creación se encargan de compilar, enlazar y desplegar el ensamblado construido en el directorio `función`.

Como alternativas, puede simplificar la tarea de construir código de rutinas .NET CLR haciéndolo en Visual Studio .NET o siguiendo manualmente los pasos de los scripts de creación de ejemplo de DB2. Consulte:

- Creación de rutinas .NET CLR (Common Language Runtime) en Visual Studio .NET
- Creación de rutinas .NET CLR (Common Language Runtime) en Visual Studio .NET utilizando la ventana de mandatos de DB2.

Los scripts de creación de ejemplo específicos del lenguaje de programación para construir rutinas .NET CLR en C# y Visual Basic se llaman `bldrtn`. Se encuentran en directorios de DB2 junto con los programas de ejemplo que se pueden crear con ellos, en las siguientes ubicaciones:

- Para C: `sqllib/samples/cs/`
- Para C++: `sqllib/samples/vb/`

Los scripts `bldrtn` se pueden utilizar para construir archivos de código fuente que contengan procedimientos y funciones definidas por el usuario. El script hace lo siguiente:

- Establece una conexión con una base de datos especificada por el usuario
- Compila y enlaza el código fuente para generar un ensamblado con un sufijo de archivo .DLL
- Copia el ensamblado en el directorio function de DB2 en el servidor de bases de datos

Los scripts `bldrtn` aceptan dos argumentos:

- El nombre de un archivo de código fuente sin sufijo de archivo
- El nombre de una base de datos con la que se establecerá una conexión

El parámetro correspondiente a la base de datos es opcional. Si no se proporciona un nombre de base de datos, el programa utiliza la base de datos por omisión sample. Puesto que las rutinas se tienen que crear en la misma instancia en la que reside la base de datos, no se necesitan argumentos para ID de usuario y contraseña.

Requisitos previos

- Hay que satisfacer los requisitos previos del sistema operativo y software de desarrollo de las rutinas .NET CLR. Consulte: "Soporte del desarrollo de rutinas .NET CLR".
- Archivo de código fuente que contenga una o más implementaciones de rutinas.
- El nombre de la base de datos dentro de la instancia de DB2 actual en la que se van a crear las rutinas.

Procedimiento

Para crear un archivo de código fuente que contenga una o más implementaciones de código de rutina, siga los pasos siguientes.

1. Abra una ventana de mandatos de DB2.
2. Copie el archivo de código fuente en el mismo directorio que el script `bldrtn`.
3. Si las rutinas se van a crear en la base de datos sample, teclee el nombre del script de creación seguido del nombre del archivo de código fuente sin la extensión de archivo .cs o .vb.

   ```
   bldrtn <nombre-archivo>
   ```

   Si las rutinas se van a crear en otra base de datos, teclee el nombre del script de creación, el nombre del archivo de código fuente sin extensión de archivo y el nombre de la base de datos:

   ```
   bldrtn <nombre-archivo> <nombre-basedatos>
   ```

   El script compila y enlaza el código fuente y genera un ensamblado. Luego, el script copia el ensamblado en el directorio de funciones del servidor de bases de datos.
4. Si no es la primera vez que se crea el archivo de código fuente que contiene las implementaciones de rutinas, detenga y vuelva a iniciar la base de datos para asegurarse de que DB2 utiliza la nueva versión de la biblioteca compartida. Lo puede hacer entrando `db2stop` seguido de `db2start` en la línea de mandatos.

Cuando haya creado satisfactoriamente la biblioteca compartida de rutinas y la haya desplegado en el directorio de función en el servidor de bases de datos, debe completar los pasos asociados a la tarea de crear rutinas C y C++.

La creación de rutinas .NET CLR incluye un paso para ejecutar la sentencia CREATE para cada rutina implementada en el archivo de código fuente. Una vez completada la creación de rutinas, puede invocar las rutinas.

Creación de código de rutina CLR (Common Language Runtime) .NET desde la ventana de mandatos de DB2

La creación de código fuente de rutinas .NET CLR es una subtarea de la creación de rutinas .NET CLR. Esta tarea se puede realizar de forma manual desde la ventana de mandatos de DB2. Se puede seguir el mismo procedimiento, independientemente de si hay o no sentencias de SQL dentro del código de la rutina. Los pasos de la tarea incluyen la compilación de código fuente escrito en un lenguaje de programación soportado por .NET CLR en un conjunto con un sufijo de archivo .DLL.

Como alternativas, puede simplificar la tarea creando código de rutinas .NET CLR haciéndolo en Visual Studio .NET o utilizando scripts de creación de ejemplo de DB2. Consulte:

- Creación de rutinas .NET CLR (Common Language Runtime) en Visual Studio .NET
- Creación de rutinas .NET CLR (Ejecución en el lenguaje común) utilizando scripts de creación de ejemplo

Requisitos previos

- Se han satisfecho los requisitos previos de sistema operativo necesario y software de desarrollo de rutinas .NET CLR. Consulte: "Soporte del desarrollo de rutinas .NET CLR".
- El código fuente se ha escrito en un lenguaje de programación de .NET CLR soportado que contiene una o más implementaciones de rutinas .NET CLR.
- El nombre de la base de datos dentro de la instancia de DB2 actual en la que se van a crear las rutinas.
- Las opciones de compilación y enlace específicas del sistema operativo necesarias para crear rutinas .NET CLR.

Para crear un archivo de código fuente que contenga una o más implementaciones de código de rutinas .NET CLR siga los pasos siguientes. A continuación se describe un ejemplo que muestra cada uno de los pasos:

1. Abra una ventana de mandatos de DB2.
2. Navegue hasta el directorio que contiene el archivo de código fuente.
3. Establezca una conexión con la base de datos en la que se van a crear las rutinas.
4. Compile el archivo de código fuente.
5. Enlace el archivo de código fuente para generar una biblioteca compartida. Para ello hay que utilizar algunas opciones de compilación y enlace específicas de DB2.
6. Copie el archivo de conjunto con el sufijo de archivo .DLL en el directorio de función de DB2 en el servidor de bases de datos.
7. Si no es la primera vez que se crea el archivo de código fuente que contiene las implementaciones de rutinas, detenga y vuelva a iniciar la base de datos para

asegurarse de que DB2 utiliza la nueva versión de la biblioteca compartida. Puede hacerlo emitiendo el mandato db2stop seguido del mandato db2start.

Cuando haya creado y desplegado satisfactoriamente la biblioteca de rutinas, debe completar los pasos asociados a la tarea de crear rutinas .NET CLR. La creación de rutinas .NET CLR incluye un paso para ejecutar la sentencia CREATE para cada rutina implementada en el archivo de código fuente. Este paso también de debe completar para poder invocar las rutinas.

Ejemplo

El siguiente ejemplo muestra la recreación de un archivo de código fuente .NET CLR. Se muestran los pasos para un archivo de código Visual Basic denominado `myVBfile.vb` que contiene implementaciones de rutinas y para un archivo de código C# denominado `myCSfile.cs`. Las rutinas se crean en un sistema operativo Windows 2000 y se utiliza Microsoft .NET Framework 1.1 para generar un conjunto de 64 bits.

1. Abra una ventana de mandatos de DB2.
2. Navegue hasta el directorio que contiene el archivo de código fuente.
3. Establezca una conexión con la base de datos en la que se van a crear las rutinas.

   ```
   db2 connect to <nombre-basedatos>
   ```

4. Compile el archivo de código fuente mediante las opciones recomendadas de compilación y enlace (donde $DB2PATH es la vía de acceso de instalación de la instancia de DB2. Sustituya este valor antes de ejecutar el mandato):

   ```
   Ejemplo en C#
   ==================
   csc /out:myCSfile.dll /target:library
       /reference:$DB2PATH%\bin\netf11\IBM.Data.DB2.dll myCSfile.cs

   Ejemplo en Visual Basic
   ====================
   vbc /target:library /libpath:$DB2PATH\bin\netf11
       /reference:$DB2PATH\bin\netf11\IBM.Data.DB2.dll
       /reference:System.dll
       /reference:System.Data.dll myVBfile.vb
   ```

   El compilador generará salida si se produce algún error. Este paso genera un archivo de exportación denominado myfile.exp.

5. Copie la biblioteca compartida en el directorio de función de DB2 en el servidor de bases de datos.

   ```
   Ejemplo en C#
   ====================
   rm -f ~HOME/sqllib/function/myCSfile.DLL
   cp myCSfile $HOME/sqllib/function/myCSfile.DLL

   Ejemplo en Visual Basic
   ====================
   rm -f ~HOME/sqllib/function/myVBfile.DLL
   cp myVBfile $HOME/sqllib/function/myVBfile.DLL
   ```

   Este paso asegura que la biblioteca de la rutina está en el directorio por omisión en el que DB2 busca bibliotecas de rutinas. Consulte el tema sobre la creación de rutinas .NET CLR para obtener más información sobre el despliegue de bibliotecas de rutinas.

6. Detenga y vuelva a iniciar la base de datos puesto que se trata de una recreación de un archivo de código fuente de rutina anteriormente creado.

```
db2stop
db2start
```

La creación de rutinas .NET CLR suele resultar más sencilla si se utilizan los scripts de creación de ejemplo específicos del sistema operativo, que también se pueden utilizar como referencia para ver cómo crear rutinas desde la línea de mandatos.

Opciones de compilación y enlace para rutinas de CLR .NET

A continuación se muestran las opciones de compilación y enlace recomendadas por DB2 para crear rutinas CLR (Common Language Runtime) .NET en Windows con el compilador Microsoft Visual Basic .NET o el compilador Microsoft C#, como se muestra en los archivos de proceso por lotes `samples\.NET\cs\bldrtn.bat` y `samples\.NET\vb\bldrtn.bat`.

Opciones de compilación y enlace para bldrtn
Opciones de compilación y enlace utilizando el compilador Microsoft C#: **csc** El compilador de Microsoft C#. **/out:%1.dll /target:library** Salida de la biblioteca de enlaces dinámicos como dll de conjunto de procedimientos almacenados. **/debug** Utilizar el depurador. **/lib: "%DB2PATH%"\bin\netf20** Utilizar la vía de acceso de bibliotecas para .NET Framework Versión 2.0. Existen varias versiones de .NET Framework a las que se da soporte para las aplicaciones: versión 1.1, versión 2.0, versión 3.0 y versión 3.5. Existe una biblioteca de enlaces dinámicos para cada una de ellas. Para .NET Framework Versión 1.1, utilice el subdirectorio `"%DB2PATH%"\bin\netf11`. Para .NET Framework Versión 2.0, 3.0 y 3.5, utilice el subdirectorio `"%DB2PATH%"\bin\netf20`. **/reference:IBM.Data.DB2.dll** Utilice la biblioteca de enlaces dinámicos de DB2 para IBM Data Server Provider para .NET. Consulte la documentación del compilador para conocer otras opciones de compilador.

Opciones de compilación y enlace utilizando el compilador Microsoft Visual Basic .NET:

vbc El compilador de Microsoft Visual Basic .NET.

/out:%1.dll /target:library
Salida de la biblioteca de enlaces dinámicos como dll de conjunto de procedimientos almacenados.

/debug Utilizar el depurador.

/libpath:"%DB2PATH%"\bin\netf20
Utilizar la vía de acceso de bibliotecas para .NET Framework Versión 2.0.

Existen varias versiones de .NET Framework a las que se da soporte para las aplicaciones: versión 1.1, versión 2.0, versión 3.0 y versión 3.5. Existe una biblioteca de enlaces dinámicos para cada una de ellas. Para .NET Framework Versión 1.1, utilice el subdirectorio `"%DB2PATH%"\bin\netf11`. Para .NET Framework Versión 2.0, 3.0 y 3.5, utilice el subdirectorio `"%DB2PATH%"\bin\netf20`.

/reference:IBM.Data.DB2.dll
Utilice la biblioteca de enlaces dinámicos de DB2 para IBM Data Server Provider para .NET.

/reference:System.dll
Referencia a la biblioteca de enlaces dinámicos de sistema Microsoft Windows.

/reference:System.Data.dll
Referencia a la biblioteca de enlaces dinámicos de datos de sistema Microsoft Windows.

Consulte la documentación del compilador para conocer otras opciones de compilador.

Depuración de rutinas .NET CLR

Es posible que sea necesario depurar rutinas .NET CLR si no se puede crear una rutina, invocar una rutina o si al invocar una rutina, ésta no se comporta o ejecuta como se esperaba.

Tome en consideración lo siguiente al depurar rutinas .NET CLR:

- Compruebe si se está utilizando un sistema operativo soportado para el desarrollo de la rutina .NET CLR.
- Compruebe si se utilizan tanto un servidor de bases de datos DB2 como un cliente DB2 soportados para el desarrollo de rutinas .NET CLR.
- Compruebe que se utiliza el software de desarrollo Microsoft .NET Framework.
- Si ha fallado la creación de la rutina:
  - Compruebe si el usuario tiene la necesaria autorización y privilegios para ejecutar la sentencia CREATE PROCEDURE o CREATE FUNCTION.
- Si ha fallado la invocación de la rutina:
  - Compruebe si el usuario tiene autorización para ejecutar la rutina. Si se ha producido un error (SQLCODE -551, SQLSTATE 42501), esto es probable ya que el invocador carece del privilegio EXECUTE sobre la rutina. El que puede otorgar este privilegio es un usuario con autorización SYSADM, DBADM o bien el definidor de la rutina.
  - Compruebe que la signatura del parámetro de rutina utilizada en la sentencia CREATE para la rutina se corresponde con la signatura del parámetro de rutina de la implementación de la rutina.

  - Compruebe que los tipos de datos utilizados en la implementación de la rutina son compatibles con los tipos de datos especificados en la signatura del parámetro de rutina de la sentencia CREATE.
  - Compruebe que en la implementación de la rutina, sean válidas las palabras clave específicas del lenguaje de .NET CLR utilizadas para indicar el método por el que debe pasarse el parámetro (por valor o referencia).
  - Compruebe que el valor especificado en la cláusula EXTERNAL de la sentencia CREATE PROCEDURE o CREATE FUNCTION se corresponde con la ubicación en la que se encuentra el ensamblaje .NET CLR que contiene la implementación de la rutina en el sistema de archivos del sistema donde está instalado el servidor de bases de datos DB2.
  - Si la rutina es una función, compruebe si todos los tipos de llamada aplicables se han programado correctamente en la implementación de la rutina. Esto es particularmente importante si la rutina se ha definido con la cláusula FINAL CALL.
- Si la rutina no se está comportando como se esperaba:
  - Modifique la rutina de modo que dé salida a la información de diagnóstico en un archivo ubicado en un directorio accesible globalmente. Dar salida a la información de diagnóstico en la pantalla no es posible desde las rutinas .NET CLR. No dirija la salida a los archivos en directorios que estén utilizados por gestores de bases de datos DB2 o bases de datos DB2.
  - Depure la rutina localmente escribiendo una simple aplicación .NET que invoque de forma directa al punto de entrada de la rutina. Para obtener información sobre cómo utilizar las características de depuración de Microsoft Visual Studio .NET, consulte la documentación del compilador Microsoft Visual Studio .NET.

Para obtener más información sobre errores comunes relacionados con la invocación y creación de rutinas .NET CLR, consulte:

- "Errores relacionados con rutinas CLR .NET"

Errores relacionados con rutinas CLR .NET

Aunque las rutinas externas comparten una implementación común en general, se pueden producir algunos errores de DB2 específicos de las rutinas CLR. Esta consulta lista los errores de CLR .NET más probables que se pueden encontrar, listados por el SQLCODE o comportamiento, con algunas sugerencias para la depuración. Los errores de DB2 relacionados con rutinas se pueden clasificar del modo siguiente:

Errores de creación de la rutina
: Errores que se producen cuando se ejecuta la sentencia CREATE para la rutina.

Errores de ejecución de la rutina
: Errores que se producen durante la invocación o ejecución de la rutina.

Independientemente de cuándo DB2 emite un error relacionado con una rutina de DB2, el texto del mensaje de error detalla la causa del error y la acción que el usuario debe emprender para resolver el problema. Hallará información adicional sobre escenarios de los errores de rutinas en el archivo de registro de diagnósticos `db2diag.log`.

Errores de creación de rutinas CLR

SQLCODE -451, SQLSTATE 42815

Este error se emite tras un intento de ejecutar una sentencia CREATE TYPE que incluye una declaración de método externo especificando la cláusula LANGUAGE con el valor CLR. No se pueden crear métodos externos de DB2 para tipos estructurados que hagan referencia a un conjunto de CLR en este momento. Cambie la cláusula LANGUAGE de forma que especifique un lenguaje soportado para el método e implemente el método en ese lenguaje alternativo.

SQLCODE -449, SQLSTATE 42878

La sentencia CREATE para la rutina CLR contiene una identificación de función o biblioteca de formato no válido en la cláusula EXTERNAL NAME. Para el lenguaje CLR, el valor de la cláusula EXTERNAL debe tomar el formato específico `'<a>:<b>!<c>'`, del modo siguiente:

- `<a>` es el archivo de conjunto de CLR en el que está ubicada la clase.
- `<b>` es la clase en la que reside el método a invocar.
- `<c>` es el método a invocar.

No se permiten caracteres en blanco iniciales ni de cola entre las comillas simples, identificadores de objeto y caracteres de separación (por ejemplo, `' <a> ! <b> '` no es válido). Sin embargo, los nombres de vía de acceso y archivo pueden contener espacios en blanco si la plataforma lo permite. Para todos los nombres de archivo, el archivo se puede especificar utilizando el formato abreviado del nombre (ejemplo: `math.dll`) o el nombre de vía de acceso completamente calificada (ejemplo: `d:\udfs\math.dll`). Si se utiliza el formato abreviado del nombre de archivo, si la plataforma es UNIX o si la rutina es una rutina LANGUAGE CLR, entonces el archivo deberá residir en el directorio de función. Si la plataforma es Windows y la rutina no es una rutina LANGUAGE CLR, el archivo debe residir en el sistema PATH. Las extensiones de archivo (ejemplos: `.a` (en UNIX), `.dll` (en Windows)) siempre se deben incluir en el nombre de archivo.

Errores de ejecución de rutinas CLR

SQLCODE -20282, SQLSTATE 42724, código de razón 1

No se ha encontrado el conjunto externo especificado por la cláusula EXTERNAL en la sentencia CREATE para la rutina.

- Compruebe si la cláusula EXTERNAL especifica el nombre correcto de conjunto de la rutina y si el conjunto se encuentra en la ubicación especificada. Si la cláusula EXTERNAL no especifica un nombre de vía de acceso completamente calificado para el conjunto deseado, DB2 supone que el nombre de vía de acceso que se proporciona es un nombre de vía de acceso relativo para el conjunto, en relación con el directorio de función de DB2.

SQLCODE -20282, SQLSTATE 42724, código de razón 2

Se ha encontrado un conjunto en la ubicación especificada por la cláusula EXTERNAL de la sentencia CREATE para la rutina, pero, en el conjunto, no se ha encontrado ninguna clase que coincida con la clase especificada en la cláusula EXTERNAL.

- Compruebe si el nombre de conjunto especificado en la cláusula EXTERNAL es el conjunto correcto para la rutina y si existe en la ubicación especificada.

- Compruebe si el nombre de clase especificado en la cláusula EXTERNAL es el nombre de clase correcto y si existe en el conjunto especificado.

SQLCODE -20282, SQLSTATE 42724, código de razón 3

Se ha encontrado un conjunto en la ubicación especificada por la cláusula EXTERNAL de la sentencia CREATE para la rutina, el cual tenía una definición de clase que coincidía correctamente, pero la signatura del método de la rutina no coincide con la signatura de la rutina especificada en la sentencia CREATE para la rutina.

- Compruebe si el nombre de conjunto especificado en la cláusula EXTERNAL es el conjunto correcto para la rutina y si existe en la ubicación especificada.
- Compruebe si el nombre de clase especificado en la cláusula EXTERNAL es el nombre de clase correcto y si existe en el conjunto especificado.
- Compruebe si la implementación del estilo de parámetro coincide con el estilo de parámetro especificado en la sentencia CREATE para la rutina.
- Compruebe si el orden de la implementación de parámetros coincide con el orden de la declaración de parámetros de la sentencia CREATE para la rutina y si se respetan los requisitos adicionales de parámetros del estilo de parámetro.
- Compruebe si los tipos de datos de los parámetros de SQL están correlacionados correctamente con los tipos de datos soportados en CLR .NET.

SQLCODE -4301, SQLSTATE 58004, código de razón 5 ó 6

Se ha producido un error al intentar el inicio o la comunicación de un intérprete de .NET. DB2 no ha podido cargar una biblioteca .NET dependiente [código de razón 5] o ha fallado una llamada al intérprete de .NET [código de razón 6].

- Asegúrese de que la instancia de DB2 está configurada correctamente para ejecutar una función o procedimiento .NET (`mscoree.dll` debe estar presente en la vía de acceso (`PATH`) del sistema). Asegúrese de que `db2clr.dll` está presente en el directorio `sqllib/bin` y de que `IBM.Data.DB2` está instalado en la antememoria del conjunto global. Si no están, asegúrese de que `.NET Framework` versión 1.1, o una versión posterior, se haya instalado en el servidor de bases de datos y de que dicho servidor esté ejecutando DB2 versión 8.2 o un release posterior.

SQLCODE -4302, SQLSTATE 38501

Se ha producido una excepción no controlada durante la ejecución, en la preparación de la ejecución o después de ejecutar la rutina. Puede ser resultado de un error de programación de la lógica de una rutina no controlado o puede ser resultado de un error de proceso interno. Para los errores de este tipo, el rastreo posterior de pila .NET que indica que la excepción no manejada se grabará en el archivo db2diag.log.

Este error también puede producirse en el caso de que la rutina haya intentado realizar una acción que no esté incluida dentro del ámbito de las acciones permitidas para la modalidad de ejecución especificada para la rutina. En este caso, se creará una entrada en db2diag.log indicando específicamente que la ha excepción se ha producido debido a una violación de control de ejecución. También se incluirá el rastreo posterior de la pila de excepción donde se ha producido la violación.

Determine si se ha comprometido o se ha modificado recientemente el conjunto de la rutina. Si la rutina se ha modificado de forma correcta, este

problema puede deberse a que la modalidad EXECUTION CONTROL de la rutina ya no esté definida en una modalidad que es adecuada para la lógica cambiada. Si está seguro de que el conjunto no se ha manipulado incorrectamente, puede modificar la modalidad de ejecución de la rutina con la sentencia ALTER PROCEDURE o ALTER FUNCTION, según sea adecuado. Para obtener más información, consulte el tema siguiente:

- "Modalidades de seguridad y de ejecución para rutinas CLR" en la página 161

Ejemplos de rutinas .NET CLR

Cuando se desarrollan rutinas .NET CLR, resulta útil consultar ejemplos para ver el aspecto que puede tener la sentencia CREATE y el código de la rutina .NET CLR. Los temas siguientes contienen ejemplos de procedimientos y funciones CLR de .NET (incluyen funciones tanto escalares como de tabla):

Procedimientos CLR de .NET

- Ejemplos de procedimientos CLR de .NET en Visual Basic
- Ejemplos de procedimientos CLR de .NET en C#

Funciones CLR de .NET

- Ejemplos de funciones CLR de .NET en Visual Basic
- Ejemplos de funciones CLR de .NET en C#

Ejemplos de procedimientos CLR de .NET en C#

Una vez comprendidos los conceptos básicos de los procedimientos, también denominados procedimientos almacenados, y los fundamentos de las rutinas Common Language Runtime .NET, puede empezar a utilizar procedimientos CLR en sus aplicaciones.

Este tema contiene ejemplos de procedimientos CLR implementados en C# que ilustran los estilos de parámetros soportados, el pase de parámetros, incluida la estructura `dbinfo`, cómo devolver un conjunto de resultados y más información. Para obtener ejemplos de UDF CLR en C#:

- "Ejemplos de funciones CLR de .NET en C#" en la página 210

Antes de trabajar con los ejemplos de procedimientos CLR, puede ser conveniente que lea los temas sobre los conceptos siguientes:

- "Rutinas CLR (Common Language Runtime) .NET" en la página 153
- "Creación de rutinas CLR .NET desde la ventana de mandatos de DB2" en la página 165
- "Creación de código de rutinas .NET CLR" en la página 167

Los ejemplos siguientes utilizan una tabla denominada `EMPLOYEE` que está incluida en la base de datos `SAMPLE`.

Utilice los ejemplos siguientes como referencias al crear sus propios procedimientos CLR en C#:

- Archivo de código externo C#
- Ejemplo 1: Procedimiento en C# del estilo de parámetro GENERAL
- Ejemplo 2: Procedimiento en C# del estilo de parámetro GENERAL WITH NULLS
- Ejemplo 3: Procedimiento en C# del estilo de parámetro SQL

- Ejemplo 4: Procedimiento en C# que devuelve un conjunto de resultados
- Ejemplo 5: Procedimiento en C# que accede a la estructura dbinfo
- Ejemplo 6: Procedimiento en C# del estilo PROGRAM TYPE MAIN

Archivo de código externo C#

Los ejemplos muestran una variedad de implementaciones de procedimientos en C#. Cada ejemplo se compone de dos partes: la sentencia CREATE PROCEDURE y la implementación en código C# externo del procedimiento desde el cual se puede crear el conjunto asociado.

El archivo fuente en C# que contiene las implementaciones de procedimientos de los ejemplos siguientes se denomina `gwenProc.cs` y tiene el formato siguiente:

Tabla 13. Formato del archivo de código externo C#

```
using System;
using System.IO;
using IBM.Data.DB2;

namespace bizLogic
{
   class empOps
   {          ...
     // C# procedures
              ...
   }
}
```

Las inclusiones del archivo se indican al principio del mismo. La inclusión `IBM.Data.DB2` es necesaria si alguno de los procedimientos del archivo contiene SQL. Existe una declaración de espacio de nombres en este archivo y una clase `empOps` que contiene los procedimientos. El uso de espacios de nombres es opcional. Si se utiliza un espacio de nombres, éste debe aparecer en el nombre de vía de acceso de ensamblaje proporcionado en la cláusula EXTERNAL de la sentencia CREATE PROCEDURE.

Es importante tener en cuenta el nombre del archivo, el espacio de nombres y el nombre de la clase, que contiene una implementación de procedimiento determinada. Estos nombres son importantes, ya que la cláusula EXTERNAL de la sentencia CREATE PROCEDURE correspondiente a cada procedimiento debe especificar esta información a fin de que DB2 pueda localizar el conjunto y la clase del procedimiento CLR.

Ejemplo 1: Procedimiento en C# del estilo de parámetro GENERAL

Este ejemplo muestra lo siguiente:

- Sentencia CREATE PROCEDURE para un procedimiento de estilo de parámetros GENERAL
- Código C# para un procedimiento del estilo de parámetro GENERAL

Este procedimiento toma un ID de empleado y una cantidad de bonificación actual como entrada. Recupera el nombre y el salario del empleado. Si la cantidad de bonificación actual es cero, se calcula una nueva bonificación basada en el salario del empleado y se devuelve junto con el nombre y apellidos del empleado. Si no se encuentra el empleado, se devuelve una serie vacía.

Tabla 14. Código para crear un procedimiento GENERAL tipo parámetro C#

```
CREATE PROCEDURE setEmpBonusGEN(IN empID CHAR(6), INOUT bonus Decimal(9,2),
                                OUT empName VARCHAR(60))
SPECIFIC SetEmpBonusGEN
LANGUAGE CLR
PARAMETER STYLE GENERAL
MODIFIES SQL DATA
EXECUTION CONTROL SAFE
FENCED
THREADSAFE
DYNAMIC RESULT SETS 0
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenProc.dll:bizLogic.empOps!SetEmpBonusGEN' ;
```

```
 public static void SetEmpBonusGEN(    String empID,
                                   ref Decimal bonus,
                                   out String empName)
 {
    // Declarar las variables locales
    Decimal salary = 0;

    DB2Command myCommand = DB2Context.GetCommand();
    myCommand.CommandText =
                    "SELECT FIRSTNME, MIDINIT, LASTNAME, SALARY "
                  + "FROM EMPLOYEE "
                  + "WHERE EMPNO = '" + empID + '";

    DB2DataReader reader = myCommand.ExecuteReader();

    if (reader.Read())  // Si se encuentra el registro de empleado
    {
       // Obtener el nombre completo y el salario del empleado
       empName = reader.GetString(0) + " " +
                 reader.GetString(1) + ". " +
                 reader.GetString(2);

       salary = reader.GetDecimal(3);

       if (bonus == 0)
       {
          if (salary > 75000)
          {
             bonus = salary * (Decimal)0.025;
          }
          else
          {
             bonus = salary * (Decimal)0.05;
          }
       }
    }
    else  // No se encuentra el empleado
    {
       empName = "";  // Establecer parámetro de salida
    }

    reader.Close();
 }
```

Ejemplo 2: Procedimiento en C# del estilo de parámetro GENERAL WITH NULLS

Este ejemplo muestra lo siguiente:

- Sentencia CREATE PROCEDURE para un procedimiento del estilo de parámetro GENERAL WITH NULLS

- Código C# para un procedimiento del estilo de parámetro GENERAL WITH NULLS

Este procedimiento toma un ID de empleado y una cantidad de bonificación actual como entrada. Si el parámetro de entrada no es nulo, recupera el nombre y salario del empleado. Si la cantidad de bonificación actual es cero, se calcula una nueva bonificación basada en el salario y se devuelve junto con el nombre y apellidos del empleado. Si no se encuentran los datos de empleado, se devuelven un entero y una serie NULL.

Tabla 15. Código para crear un procedimiento en C# del estilo de parámetro GENERAL WITH NULLS

```
CREATE PROCEDURE SetEmpbonusGENNULL(IN empID CHAR(6),
                                    INOUT bonus Decimal(9,2),
                                    OUT empName VARCHAR(60))
SPECIFIC SetEmpbonusGENNULL
LANGUAGE CLR
PARAMETER STYLE GENERAL WITH NULLS
DYNAMIC RESULT SETS 0
MODIFIES SQL DATA
EXECUTION CONTROL SAFE
FENCED
THREADSAFE
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenProc.dll:bizLogic.empOps!SetEmpBonusGENNULL'
;
```

Tabla 15. Código para crear un procedimiento en C# del estilo de parámetro GENERAL WITH NULLS (continuación)

```
public static void SetEmpBonusGENNULL(    String empID,
                                       ref Decimal bonus,
                                       out String empName,
                                           Int16[] NullInds)
{
   Decimal salary = 0;
   if (NullInds[0] == -1) // Comprobar si la entrada es nula
   {
     NullInds[1] = -1;    // Devolver un valor de bonificación NULL
     empName = "";        // Establecer el valor de salida
     NullInds[2] = -1;    // Devolver un valor empName NULL
   }
   else
   {
      DB2Command myCommand = DB2Context.GetCommand();
       myCommand.CommandText =
                     "SELECT FIRSTNME, MIDINIT, LASTNAME, SALARY "
                   + "FROM EMPLOYEE "
                   + "WHERE EMPNO = '" + empID + "'";
      DB2DataReader reader = myCommand.ExecuteReader();

      if (reader.Read())  // Si se encuentra el registro de empleado
      {
         // Obtener el nombre completo y el salario del empleado
         empName = reader.GetString(0) + " "
         +
                   reader.GetString(1) + ". " +
                   reader.GetString(2);
         salary = reader.GetDecimal(3);

         if (bonus == 0)
         {
           if (salary > 75000)
           {
              bonus = salary * (Decimal)0.025;
              NullInds[1] = 0; // Devolver un valor no NULL
           }
           else
           {
              bonus = salary * (Decimal)0.05;
              NullInds[1] = 0; // Devolver un valor no NULL
           }
         }
      }
      else  // No se encuentra el empleado
      {
         empName = "*sdq;;         // Establecer parámetro de salida
         NullInds[2] = -1;     // Devolver un valor NULL
      }

      reader.Close();
   }
}
```

Ejemplo 3: Procedimiento en C# del estilo de parámetro SQL

Este ejemplo muestra lo siguiente:

- Sentencia CREATE PROCEDURE para un procedimiento del estilo de parámetros SQL
- Código C# para un procedimiento del estilo de parámetro SQL

Este procedimiento toma un ID de empleado y una cantidad de bonificación actual como entrada. Recupera el nombre y el salario del

empleado. Si la cantidad de bonificación actual es cero, se calcula una nueva bonificación basada en el salario y se devuelve junto con el nombre y apellidos del empleado. Si no se encuentra el empleado, se devuelve una serie vacía.

Tabla 16. Creación de un procedimiento en C# del estilo de parámetro de SQL con parámetros

```
CREATE PROCEDURE SetEmpbonusSQL(IN empID CHAR(6),
                                INOUT bonus Decimal(9,2),
                                OUT empName VARCHAR(60))
SPECIFIC SetEmpbonusSQL
LANGUAGE CLR
PARAMETER STYLE SQL
DYNAMIC RESULT SETS 0
MODIFIES SQL DATA
FENCED
THREADSAFE
EXECUTION CONTROL SAFE
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenProc.dll:bizLogic.empOps!SetEmpBonusSQL' ;
```

Tabla 16. Creación de un procedimiento en C# del estilo de parámetro de SQL con parámetros (continuación)

```
public static void SetEmpBonusSQL(    String empID,
                                   ref Decimal bonus,
                                   out String empName,
                                       Int16  empIDNullInd,
                                   ref Int16  bonusNullInd,
                                   out Int16  empNameNullInd,
                                   ref string sqlStateate,
                                       string funcName,
                                       string specName,
                                   ref string sqlMessageText)
{
   // Declarar las variables locales del lenguaje principal
   Decimal salary eq; 0;

   if (empIDNullInd == -1) // Comprobar si la entrada es nula
   {
      bonusNullInd = -1;   // Devolver un valor de bonificación NULL
      empName = "";
      empNameNullInd = -1; // Devolver un valor empName NULL
   }
   else
   {
      DB2Command myCommand = DB2Context.GetCommand();
      myCommand.CommandText =
                        "SELECT FIRSTNME, MIDINIT, LASTNAME, SALARY
                        "
                     + "FROM EMPLOYEE "
                     + "WHERE EMPNO = '" + empID + "'";

      DB2DataReader reader = myCommand.ExecuteReader();

      if (reader.Read())  // Si se encuentra el registro de empleado
      {
         // Obtener el nombre completo y el salario del empleado
         empName = reader.GetString(0) + " "
         +
         reader.GetString(1) + ". " +
         reader.GetString(2);
         empNameNullInd = 0;
         salary = reader.GetDecimal(3);

         if (bonus == 0)
         {
            if (salary > 75000)
            {
               bonus = salary * (Decimal)0.025;
               bonusNullInd = 0;  // Devolver un valor distinto de NULL
            }
            else
            {
               bonus = salary * (Decimal)0.05;
               bonusNullInd = 0;  // Devolver un valor distinto de NULL
            }
         }
      }
      else  // No se encuentra el empleado
      {
         empName = "";         // Establecer parámetro de salida
         empNameNullInd = -1;  // Devolver un valor NULL
      }
      reader.Close();
   }
}
```

Ejemplo 4: Procedimiento en C# del estilo de parámetro GENERAL que devuelve un conjunto de resultados

Este ejemplo muestra lo siguiente:

- Sentencia CREATE PROCEDURE para un procedimiento en C# externo que devuelve un conjunto de resultados
- Código C# para un procedimiento del estilo de parámetro GENERAL que devuelve un conjunto de resultados

Este procedimiento acepta el nombre de una tabla como parámetro. Devuelve un conjunto de resultados que contiene todas las filas de la tabla especificada por el parámetro de entrada. Esto se realiza dejando abierto un `DB2DataReader` para un conjunto de resultados de consulta determinado cuando el procedimiento efectúa la devolución. Específicamente, si no se ejecuta `reader.Close()`, se devolverá el conjunto de resultados.

Tabla 17. Código para crear un procedimiento en C# que devuelve un conjunto de resultados

```
  CREATE PROCEDURE ReturnResultSet(IN tableName VARCHAR(20))
SPECIFIC ReturnResultSet
  DYNAMIC RESULT SETS 1
  LANGUAGE CLR
  PARAMETER STYLE GENERAL
  FENCED
  PROGRAM TYPE SUB
  EXTERNAL NAME 'gwenProc.dll:bizLogic.empOps!ReturnResultSet' ;
```

```
   public static void ReturnResultSet(string tableName)
   {
      DB2Command myCommand = DB2Context.GetCommand();

      // Establecer la sentencia de SQL a ejecutar y ejecutarla.
      myCommand.CommandText = "SELECT * FROM " + tableName;
      DB2DataReader reader = myCommand.ExecuteReader();

      // El DB2DataReader contiene el resultado de la consulta.
      // Este conjunto de resultados se puede devolver con el
      // procedimiento simplemente NO cerrando el DB2DataReader.
      // Específicamente, NO ejecutar reader.Close();
   }
```

Ejemplo 5: Procedimiento en C# del estilo de parámetro SQL que accede a la estructura dbinfo

Este ejemplo muestra lo siguiente:

- Sentencia CREATE PROCEDURE para un procedimiento que accede a la estructura `dbinfo`
- Código C# para un procedimiento del estilo de parámetro SQL que accede a la estructura `dbinfo`

Para acceder a la estructura `dbinfo`, se debe especificar la cláusula DBINFO en la sentencia CREATE PROCEDURE. No es necesario ningún parámetro para la estructura `dbinfo` en la sentencia CREATE PROCEDURE; no obstante, se debe crear un parámetro para la misma en el código externo de la rutina. Este procedimiento sólo devuelve el valor del nombre de base de datos actual desde el campo `dbname` de la estructura `dbinfo`.

Tabla 18. Código para crear un procedimiento en C# que accede a la estructura dbinfo

```
CREATE PROCEDURE ReturnDbName(OUT dbName VARCHAR(20))
SPECIFIC ReturnDbName
DYNAMIC RESULT SETS 0
LANGUAGE CLR
PARAMETER STYLE SQL
FENCED
THREADSAFE
EXECUTION CONTROL SAFE
DBINFO
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenProc.dll:bizLogic.empOps!ReturnDbName'
;
```

```
 public static void ReturnDbName(out string dbName,
                                  out Int16  dbNameNullInd,
                                  ref string sqlState,
                                      string funcName,
                                      string specName,
                                  ref string sqlMessageText,
                                      sqludf_dbinfo dbinfo)
 {
    // Recuperar el nombre de base de datos actual desde la
    // estructura dbinfo y devolverlo.
    // ** Nota ** Los nombres del campo dbinfo son sensibles
    // a las mayúsculas y minúsculas
    dbName = dbinfo.dbname;
    dbNameNullInd = 0;  // Devolver un valor no nulo;

    // Si desea devolver un error definido por el usuario en la
    // SQLCA puede especificar un sqlStateate de 5 dígitos definido
    // por el usuario y el texto de la serie de un mensaje de error.
    // Por ejemplo:
    //
    //   sqlStateate = "ABCDE";
    //   sqlMessageText = "Se ha producido un error definido por el usuario"
    //
    //  DB2 devuelve los valores anteriores al cliente en la
    //  estructura SQLCA. Los valores se utilizan para generar
    //  un error sqlStateate estándar de DB2.
 }
```

Ejemplo 6: Procedimiento en C# con el estilo PROGRAM TYPE MAIN

Este ejemplo muestra lo siguiente:

- Sentencia CREATE PROCEDURE para un procedimiento que utiliza un estilo de programa principal
- Código C# para el estilo de parámetro GENERAL WITH NULLS en la utilización de un estilo de programa MAIN

Para implementar una rutina en un estilo de programa principal, se debe especificar la cláusula PROGRAM TYPE en la sentencia CREATE PROCEDURE con el valor MAIN. Se especifican parámetros en la sentencia CREATE PROCEDURE; no obstante, en la implementación del código, los parámetros se pasan a la rutina en un parámetro entero `argc` y una matriz de parámetros `argv`.

Tabla 19. Código para crear un procedimiento en C# del estilo PROGRAM TYPE MAIN

```
CREATE PROCEDURE MainStyle( IN empID CHAR(6),
                            INOUT bonus Decimal(9,2),
                            OUT empName VARCHAR(60))
SPECIFIC MainStyle
DYNAMIC RESULT SETS 0
LANGUAGE CLR
PARAMETER STYLE GENERAL WITH NULLS
MODIFIES SQL DATA
FENCED
THREADSAFE
EXECUTION CONTROL SAFE
PROGRAM TYPE MAIN
EXTERNAL NAME 'gwenProc.dll:bizLogic.empOps!main' ;
```

Tabla 19. Código para crear un procedimiento en C# del estilo PROGRAM TYPE MAIN (continuación)

```
public static void main(Int32 argc, Object[]
argv)
{
  String empID = (String)argv[0];   // argv[0] has nullInd:argv[3]
  Decimal bonus = (Decimal)argv[1]; // argv[1] has nullInd:argv[4]
                                    // argv[2] has nullInd:argv[5]
  Decimal salary = 0;
  Int16[] NullInds = (Int16[])argv[3];

  if ((NullInds[0]) == (Int16)(-1)) // Comprobar si empID es nulo
  {
     NullInds[1] = (Int16)(-1);  // Devolver un valor de bonificación NULL
     argv[1] = (String)"";       // Establecer parámetro de salida empName
     NullInds[2] = (Int16)(-1);  // Devolver un valor empName NULL
     Return;
  }
  else
  {
     DB2Command myCommand = DB2Context.GetCommand();
     myCommand.CommandText =
                      "SELECT FIRSTNME, MIDINIT, LASTNAME, salary "
                   + "FROM EMPLOYEE "
                   + "WHERE EMPNO = '" + empID + "'";

     DB2DataReader reader = myCommand.ExecuteReader();

     if (reader.Read())  // Si se encuentra el registro de empleado
     {
       // Obtener el nombre completo y el salario del empleado
       argv[2] = (String) (reader.GetString(0) + " " +
                           reader.GetString(1) + ".
                           " +
                           reader.GetString(2));
       NullInds[2] = (Int16)0;
       salary = reader.GetDecimal(3);

       if (bonus == 0)
       {
          if (salary > 75000)
          {
             argv[1] = (Decimal)(salary * (Decimal)0.025);
             NullInds[1] = (Int16)(0);  // Devolver un valor no NULL
          }
          else
          {
             argv[1] = (Decimal)(salary * (Decimal)0.05);
             NullInds[1] = (Int16)(0);  // Devolver un valor no NULL
          }
       }
     }
     else  // No se encuentra el empleado
     {
        argv[2] = (String)("");         // Establecer parámetro de salida
        NullInds[2] = (Int16)(-1);      // Devolver un valor NULL
     }

     reader.Close();
   }
}
```

Ejemplos de funciones CLR de .NET en Visual Basic

Una vez comprendidos los conceptos básicos de las funciones definidas por el usuario (UDF) y los fundamentos de las rutinas CLR, puede empezar a explotar las UDF CLR en sus aplicaciones y en el entorno de bases de datos. Este tema contiene algunos ejemplos de UDF CLR para poder empezar. Si desea obtener ejemplos de procedimientos CLR en Visual Basic:

- "Ejemplos de procedimientos CLR de .NET en Visual Basic" en la página 192

Antes de trabajar con los ejemplos de UDF CLR, puede ser conveniente que lea los temas sobre los conceptos siguientes:

- "Rutinas CLR (Common Language Runtime) .NET" en la página 153
- "Creación de rutinas CLR .NET desde la ventana de mandatos de DB2" en la página 165
- "Funciones escalares externas" en la página 139
- "Creación de código de rutinas .NET CLR" en la página 167

Los ejemplos siguientes utilizan una tabla denominada `EMPLOYEE` que está incluida en la base de datos `SAMPLE`.

Utilice los ejemplos siguientes como referencias al crear sus propias UDF CLR en Visual Basic:

- Archivo de código externo Visual Basic
- Ejemplo 1: Función de tabla en Visual Basic del estilo de parámetro SQL
- Ejemplo 2: Función escalar en Visual Basic del estilo de parámetro SQL

Archivo de código externo Visual Basic

Los ejemplos siguientes muestran una variedad de implementaciones de UDF en Visual Basic. La sentencia CREATE FUNCTION se proporciona para cada UDF con el código fuente Visual Basic correspondiente desde el cual se puede crear el conjunto asociado. El archivo fuente en Visual Basic que contiene las declaraciones de funciones utilizadas en los ejemplos siguientes se denomina `gwenVbUDF.cs` y tiene el formato siguiente:

Tabla 20. Formato del archivo de código externo Visual Basic

```
using System;
using System.IO;
using IBM.Data.DB2;

Namespace bizLogic

   ...
   ' Definiciones de clases que contienen declaraciones de UDF
   ' y cualquier definición de clase de soporte
   ...

End Namespace
```

Las declaraciones de funciones deben estar incluidas en una clase dentro de un archivo de Visual Basic. El uso de espacios de nombres es opcional. Si se utiliza un espacio de nombres, éste debe aparecer en el nombre de vía de acceso de conjunto proporcionado en la cláusula EXTERNAL de la sentencia CREATE PROCEDURE. La inclusión de `IBM.Data.DB2.` es necesaria si la función contiene SQL.

Ejemplo 1: Función de tabla en Visual Basic del estilo de parámetro SQL

Este ejemplo muestra lo siguiente:

- Sentencia CREATE FUNCTION para una función de tabla del estilo de parámetro SQL
- Código Visual Basic para una función de tabla del estilo de parámetro SQL

Esta función de tabla devuelve una tabla que contiene filas de los datos de empleado que se han creado a partir de una matriz de datos. Existen dos clases asociadas con este ejemplo. La clase `person` representa los empleados y la clase `empOps` contiene la UDF de tabla de rutina que utiliza la clase `person`. La información sobre el salario de los empleados se actualiza basándose en el valor de un parámetro de entrada. La matriz de datos de este ejemplo se crea dentro de la propia función de tabla en la primera llamada de la función de tabla. Dicha matriz también se podría haber creado leyendo datos de un archivo de texto del sistema de archivos. Los valores de datos de la matriz se graban en un área reutilizable para que sea posible acceder a los datos en llamadas subsiguientes de la función de tabla.

En cada llamada de la función de tabla, se lee un registro de la matriz y se genera una fila en la tabla devuelta por la función. La fila se genera en la tabla estableciendo los parámetros de salida de la función de tabla en los valores de fila deseados. Después de que se produzca la llamada final de la función de tabla, se devuelve la tabla de las filas generadas.

Tabla 21. Código para crear una función de tabla en Visual Basic del estilo de parámetro SQL

```
CREATE FUNCTION TableUDF(double)
RETURNS TABLE (name varchar(20),
               job varchar(20),
               salary double)
EXTERNAL NAME 'gwenVbUDF.dll:bizLogic.empOps!TableUDF'
LANGUAGE CLR
PARAMETER STYLE SQL
NOT DETERMINISTIC
FENCED
SCRATCHPAD 10
FINAL CALL
DISALLOW PARALLEL
NO DBINFO
EXECUTION CONTROL SAFE
```

Tabla 21. Código para crear una función de tabla en Visual Basic del estilo de parámetro SQL (continuación)

```
Class Person
' La clase Person es una clase de soporte para
' la función de tabla UDF, tableUDF, siguiente.

  Private name As String
  Private position As String
  Private salary As Int32

  Public Sub New(ByVal newName As String, _
                 ByVal newPosition As String, _
                 ByVal newSalary As Int32)

    name = newName
    position = newPosition
    salary = newSalary
  End Sub

  Public Property GetName() As String
    Get
      Return name
    End Get

    Set (ByVal value As String)
      name = value
    End Set
  End Property

  Public Property GetPosition() As String
    Get
      Return position
    End Get

    Set (ByVal value As String)
      position = value
    End Set
  End Property

  Public Property GetSalary() As Int32
    Get
      Return salary
    End Get

    Set (ByVal value As Int32)
      salary = value
    End Set
  End Property

End Class
```

Tabla 21. Código para crear una función de tabla en Visual Basic del estilo de parámetro SQL (continuación)

```
Class empOps

  Public Shared Sub TableUDF(byVal factor as Double, _
                             byRef name As String, _
                             byRef position As String, _
                             byRef salary As Double, _
                             byVal factorNullInd As Int16, _
                             byRef nameNullInd As Int16, _
                             byRef positionNullInd As Int16, _
                             byRef salaryNullInd As Int16, _
                             byRef sqlState As String, _
                             byVal funcName As String, _
                             byVal specName As String, _
                             byRef sqlMessageText As String, _
                             byVal scratchPad As Byte(), _
                             byVal callType As Int32)

    Dim intRow As Int16

    intRow = 0

    ' Crear una matriz de información del tipo Person
    Dim staff(2) As Person
    staff(0) = New Person("Gwen", "Developer", 10000)
    staff(1) = New Person("Andrew", "Developer", 20000)
    staff(2) = New Person("Liu", "Team Leader", 30000)

    ' Inicializar valores de parámetro de salida e indicadores NULL
    salary = 0
    name = position = ""
    nameNullInd = positionNullInd = salaryNullInd = -1

    Select callType
       Case -2   ' Case SQLUDF_TF_FIRST:
       Case -1   ' Case SQLUDF_TF_OPEN:
         intRow = 1
         scratchPad(0) = intRow  ' Grabar en área reutilizable
       Case 0    ' Case SQLUDF_TF_FETCH:
         intRow = scratchPad(0)
         If intRow > staff.Length
            sqlState = "02000"  ' Devolver un error SQLSTATE
         Else
            ' Generar una fila en la tabla de salida
            ' basada en los datos de la matriz staff.
            name = staff(intRow).GetName()
            position = staff(intRow).GetPosition()
            salary = (staff(intRow).GetSalary()) * factor
            nameNullInd = 0
            positionNullInd = 0
            salaryNullInd = 0
         End If
         intRow = intRow + 1
         scratchPad(0) = intRow  ' Grabar área reutilizable

       Case 1    ' Case SQLUDF_TF_CLOSE:

       Case 2    ' Case SQLUDF_TF_FINAL:
    End Select

  End Sub

End Class
```

Ejemplo 2: Función escalar en Visual Basic del estilo de parámetro SQL

Este ejemplo muestra lo siguiente:

- Sentencia CREATE FUNCTION para una función escalar del estilo de parámetro SQL
- Código Visual Basic para una función escalar del estilo de parámetro SQL

Esta función escalar devuelve un solo valor de cuenta para cada valor de entrada sobre el que actúa. Para un valor de entrada situado en la n[a] posición del conjunto de valores de entrada, el valor escalar de salida es el valor n. En cada llamada de la función escalar, donde una llamada está asociada con cada fila o valor del conjunto de filas o valores de entrada, la cuenta aumenta en uno y se devuelve el valor actual de la cuenta. Luego, la cuenta se guarda en el almacenamiento intermedio de memoria del área reutilizable para mantener el valor de cuenta entre cada llamada de la función escalar.

Esta función escalar se puede invocar fácilmente si, por ejemplo, se dispone de una tabla definida del modo siguiente:

```
CREATE TABLE T (i1 INTEGER);
INSERT INTO T VALUES 12, 45, 16, 99;
```

Se puede utilizar una consulta simple como la siguiente para invocar la función escalar:

```
SELECT my_count(i1) as count, i1 FROM T;
```

La salida de una consulta como la indicada sería:

```
COUNT           I1
-----------     ----------
1               12
2               45
3               16
4               99
```

Esta UDF escalar es bastante simple. En lugar de devolver sólo la cuenta de las filas, puede utilizar una función escalar que formatee los datos de una columna existente. Por ejemplo, puede añadir una serie a cada valor de una columna de direcciones, puede crear una serie compleja a partir de una cadena de series de entrada o puede efectuar un cálculo matemático complejo con un conjunto de datos donde deberá almacenar un resultado intermedio.

Tabla 22. Código para crear una función escalar en Visual Basic del estilo de parámetro SQL

```
CREATE FUNCTION mycount(INTEGER)
RETURNS INTEGER
LANGUAGE CLR
PARAMETER STYLE SQL
NO SQL
SCRATCHPAD 10
FINAL CALL
FENCED
EXECUTION CONTROL SAFE
NOT DETERMINISTIC
EXTERNAL NAME 'gwenUDF.dll:bizLogic.empOps!CountUp';
```

Tabla 22. Código para crear una función escalar en Visual Basic del estilo de parámetro SQL (continuación)

```
Class empOps
  Public Shared Sub CountUp(byVal input As Int32, _
                            byRef outCounter As Int32, _
                            byVal nullIndInput As Int16, _
                            byRef nullIndOutCounter As Int16, _
                            byRef sqlState As String, _
                            byVal qualName As String, _
                            byVal specName As String, _
                            byRef sqlMessageText As String, _
                            byVal scratchPad As Byte(), _
                            byVal callType As Int32)

     Dim counter As Int32
     counter = 1

     Select callType
        case -1           ' case SQLUDF_TF_OPEN_CALL
           scratchPad(0) = counter
           outCounter = counter
           nullIndOutCounter = 0
        case 0              'case SQLUDF_TF_FETCH_CALL:
           counter = scratchPad(0)
           counter = counter + 1
           outCounter = counter
           nullIndOutCounter = 0
           scratchPad(0) = counter
        case 1             'case SQLUDF_CLOSE_CALL:
           counter = scratchPad(0)
           outCounter = counter
           nullIndOutCounter = 0
        case Else          ' Nunca se debe entrar aquí
           ' Estos casos no se producirán por las razones siguientes:
           ' Case -2  (SQLUDF_TF_FIRST)     ->No hay FINAL CALL en sent. CREATE
           ' Case 2   (SQLUDF_TF_FINAL)     ->No hay FINAL CALL en sent. CREATE
           ' Case 255 (SQLUDF_TF_FINAL_CRA) ->No se utiliza SQL en la función
           '
           ' * Nota*
           ' ---------
           ' Else es necesario para que durante la compilación
           ' se establezca siempre el parámetro de salida outCounter *
           outCounter = 0
           nullIndOutCounter = -1
     End Select
  End Sub

End Class
```

Ejemplos de procedimientos CLR de .NET en Visual Basic

Una vez comprendidos los conceptos básicos de los procedimientos, también denominados procedimientos almacenados, y los fundamentos de las rutinas de ejecución en el lenguaje común .NET, puede empezar a utilizar procedimientos CLR en sus aplicaciones.

Este tema contiene ejemplos de procedimientos CLR implementados en `Visual Basic`; éstos ilustran los estilos de parámetros soportados, el pase de parámetros, incluida la estructura `dbinfo`, cómo devolver un conjunto de resultados y más información. Para obtener ejemplos de UDF CLR en `Visual Basic`:

- "Ejemplos de funciones CLR de .NET en Visual Basic" en la página 187

Antes de trabajar con los ejemplos de procedimientos CLR, puede ser conveniente que lea los temas sobre los conceptos siguientes:

- "Rutinas CLR (Common Language Runtime) .NET" en la página 153
- "Creación de rutinas CLR .NET desde la ventana de mandatos de DB2" en la página 165
- "Beneficios del uso de rutinas" en la página 2
- "Creación de código de rutinas .NET CLR" en la página 167

Los ejemplos siguientes utilizan una tabla denominada `EMPLOYEE` que está incluida en la base de datos `SAMPLE`.

Utilice los ejemplos siguientes como referencias al crear sus propios procedimientos CLR en Visual Basic:

- Archivo de código externo Visual Basic
- Ejemplo 1: Procedimiento en Visual Basic del estilo de parámetros GENERAL
- Ejemplo 2: Procedimiento en Visual Basic del estilo de parámetros GENERAL WITH NULLS
- Ejemplo 3: Procedimiento en Visual Basic del estilo de parámetros SQL
- Ejemplo 4: Procedimiento en Visual Basic que devuelve un conjunto de resultados
- Ejemplo 5: Procedimiento en Visual Basic que accede a la estructura dbinfo
- Ejemplo 6: Procedimiento en Visual Basic del estilo PROGRAM TYPE MAIN

Archivo de código externo Visual Basic

Los ejemplos muestran una variedad de implementaciones de procedimientos en Visual Basic. Cada ejemplo se compone de dos partes: la sentencia CREATE PROCEDURE y la implementación en código Visual Basic externo del procedimiento desde el cual se puede crear el conjunto asociado.

El archivo fuente en Visual Basic que contiene las implementaciones de procedimientos de los ejemplos siguientes se denomina `gwenVbProc.vb` y tiene el formato siguiente:

Tabla 23. Formato del archivo de código externo Visual Basic

```
using System;
using System.IO;
using IBM.Data.DB2;

Namespace bizLogic

   Class empOps
             ...
     ' Procedimientos en Visual Basic
             ...
   End Class
End Namespace
```

Las inclusiones del archivo se indican al principio del mismo. La inclusión `IBM.Data.DB2` es necesaria si alguno de los procedimientos del archivo contiene SQL. Existe una declaración de espacio de nombres en este archivo y una clase `empOps` que contiene los procedimientos. El uso de espacios de nombres es opcional. Si se utiliza un espacio de nombres, éste debe aparecer en el nombre de vía de acceso de ensamblaje proporcionado en la cláusula EXTERNAL de la sentencia CREATE PROCEDURE.

Es importante tener en cuenta el nombre del archivo, el espacio de nombres y el nombre de la clase, que contiene una implementación de procedimiento determinada. Estos nombres son importantes, ya que la cláusula EXTERNAL de la sentencia CREATE PROCEDURE correspondiente a cada procedimiento debe especificar esta información a fin de que DB2 pueda localizar el conjunto y la clase del procedimiento CLR.

Ejemplo 1: Procedimiento en Visual Basic del estilo de parámetros GENERAL

Este ejemplo muestra lo siguiente:

- Sentencia CREATE PROCEDURE para un procedimiento de estilo de parámetros GENERAL
- Código Visual Basic para un procedimiento del estilo de parámetros GENERAL

Este procedimiento toma un ID de empleado y una cantidad de bonificación actual como entrada. Recupera el nombre y el salario del empleado. Si la cantidad de bonificación actual es cero, se calcula una nueva bonificación basada en el salario del empleado y se devuelve junto con el nombre y apellidos del empleado. Si no se encuentra el empleado, se devuelve una serie vacía.

Tabla 24. Código para crear un procedimiento en Visual Basic del estilo de parámetros GENERAL

```
CREATE PROCEDURE SetEmpBonusGEN(IN empId CHAR(6),
                                INOUT bonus Decimal(9,2),
                                OUT empName VARCHAR(60))
SPECIFIC setEmpBonusGEN
LANGUAGE CLR
PARAMETER STYLE GENERAL
DYNAMIC RESULT SETS 0
FENCED
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenVbProc.dll:bizLogic.empOps!SetEmpBonusGEN'
```

Tabla 24. Código para crear un procedimiento en Visual Basic del estilo de parámetros GENERAL (continuación)

```
Public Shared Sub SetEmpBonusGEN(ByVal empId As String, _
                                 ByRef bonus As Decimal, _
                                 ByRef empName As String)

   Dim salary As Decimal
   Dim myCommand As DB2Command
   Dim myReader As DB2DataReader

   salary = 0

   myCommand = DB2Context.GetCommand()
   myCommand.CommandText = _
             "SELECT FIRSTNME, MIDINIT, LASTNAME, SALARY " _
          + "FROM EMPLOYEE " _
          + "WHERE EMPNO = '" + empId + "'"
   myReader = myCommand.ExecuteReader()

   If myReader.Read()  ' Si se encuentra el registro de empleado
      ' Obtener nombre y apellidos y salario del empleado
      empName = myReader.GetString(0) + " "  _
              + myReader.GetString(1) + ". " _
              + myReader.GetString(2)

      salary = myReader.GetDecimal(3)

      If bonus = 0
         If salary > 75000
            bonus = salary * 0.025
         Else
            bonus = salary * 0.05
         End If
      End If
   Else  ' No se encuentra el empleado
      empName = ""            ' Establecer el parámetro de salida
   End If

   myReader.Close()

End Sub
```

Ejemplo 2: Procedimiento en Visual Basic del estilo de parámetros GENERAL WITH NULLS

Este ejemplo muestra lo siguiente:

- Sentencia CREATE PROCEDURE para un procedimiento del estilo de parámetro GENERAL WITH NULLS
- Código Visual Basic para un procedimiento del estilo de parámetros GENERAL WITH NULLS

Este procedimiento toma un ID de empleado y una cantidad de bonificación actual como entrada. Si el parámetro de entrada no es nulo, recupera el nombre y salario del empleado. Si la cantidad de bonificación actual es cero, se calcula una nueva bonificación basada en el salario y se devuelve junto con el nombre y apellidos del empleado. Si no se encuentran los datos de empleado, se devuelven un entero y una serie NULL.

Tabla 25. Código para crear un procedimiento en Visual Basic del estilo de parámetros GENERAL WITH NULLS

```
CREATE PROCEDURE SetEmpBonusGENNULL(IN empId CHAR(6),
                                    INOUT bonus Decimal(9,2),
                                    OUT empName VARCHAR(60))
SPECIFIC SetEmpBonusGENNULL
LANGUAGE CLR
PARAMETER STYLE GENERAL WITH NULLS
DYNAMIC RESULT SETS 0
FENCED
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenVbProc.dll:bizLogic.empOps!SetEmpBonusGENNULL'
```

```
  Public Shared Sub SetEmpBonusGENNULL(ByVal empId As String, _
                                       ByRef bonus As Decimal, _
                                       ByRef empName As String, _
                                       byVal nullInds As Int16())

     Dim salary As Decimal
     Dim myCommand As DB2Command
     Dim myReader As DB2DataReader

     salary = 0

     If nullInds(0) = -1   ' Comprobar si la entrada es nula
        nullInds(1) = -1   ' Devolver un valor de bonificación NULL
        empName = ""            ' Establecer el parámetro de salida
        nullInds(2) = -1   ' Devolver un valor empName NULL
        Return
     Else
        myCommand = DB2Context.GetCommand()
        myCommand.CommandText = _
                  "SELECT FIRSTNME, MIDINIT, LASTNAME, SALARY " _
                + "FROM EMPLOYEE " _
                + "WHERE EMPNO = '" + empId + "'"

        myReader = myCommand.ExecuteReader()

        If myReader.Read()  ' Si se encuentra el registro de empleado
           ' Obtener nombre y apellidos y salario del empleado
           empName = myReader.GetString(0) + " "  _
                   + myReader.GetString(1) + ". " _
                + myReader.GetString(2)

           salary = myReader.GetDecimal(3)

           If bonus = 0
              If salary > 75000
                 bonus = Salary * 0.025
                 nullInds(1) = 0  'Devolver un valor distinto de NULL
              Else
                 bonus = salary * 0.05
                 nullInds(1) = 0  ' Devolver un valor distinto de NULL
              End If
           Else  'No se encuentra el empleado
              empName = ""            ' Establecer el parámetro de salida
              nullInds(2) = -1    ' Devolver un valor NULL
           End If
        End If

        myReader.Close()

     End If

  End Sub
```

Ejemplo 3: Procedimiento en Visual Basic del estilo de parámetro SQL

Este ejemplo muestra lo siguiente:

- Sentencia CREATE PROCEDURE para un procedimiento del estilo de parámetro SQL
- Código Visual Basic para un procedimiento del estilo de parámetros SQL

Este procedimiento toma un ID de empleado y una cantidad de bonificación actual como entrada. Recupera el nombre y el salario del empleado. Si la cantidad de bonificación actual es cero, se calcula una nueva bonificación basada en el salario y se devuelve junto con el nombre y apellidos del empleado. Si no se encuentra el empleado, se devuelve una serie vacía.

Tabla 26. Código para crear un procedimiento en Visual Basic del estilo de parámetros SQL con parámetros

```
CREATE PROCEDURE SetEmpBonusSQL(IN empId CHAR(6),
                                INOUT bonus Decimal(9,2),
                                OUT empName VARCHAR(60))
SPECIFIC SetEmpBonusSQL
LANGUAGE CLR
PARAMETER STYLE SQL
DYNAMIC RESULT SETS 0
FENCED
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenVbProc.dll:bizLogic.empOps!SetEmpBonusSQL'
```

Tabla 26. Código para crear un procedimiento en Visual Basic del estilo de parámetros SQL con parámetros (continuación)

```
Public Shared Sub SetEmpBonusSQL(byVal empId As String, _
                                 byRef bonus As Decimal, _
                                 byRef empName As String, _
                                 byVal empIdNullInd As Int16, _
                                 byRef bonusNullInd As Int16, _
                                 byRef empNameNullInd As Int16, _
                                 byRef sqlState As String, _
                                 byVal funcName As String, _
                                 byVal specName As String, _
                                 byRef sqlMessageText As String)

   ' Declarar las variables locales del lenguaje principal
   Dim salary As Decimal
   Dim myCommand As DB2Command
   Dim myReader As DB2DataReader

   salary = 0

   If empIdNullInd = -1   ' Comprobar si la entrada es nula
      bonusNullInd = -1   ' Devolver un valor de bonificación NULL
      empName = ""
      empNameNullInd = -1 ' Devolver un valor empName NULL
   Else
     myCommand = DB2Context.GetCommand()
     myCommand.CommandText = _
                 "SELECT FIRSTNME, MIDINIT, LASTNAME, SALARY " _
              + "FROM EMPLOYEE " _
              + " WHERE EMPNO = '" + empId + "'"

      myReader = myCommand.ExecuteReader()

      If myReader.Read()  ' Si se encuentra el registro de empleado
         ' Obtener nombre y apellidos y salario del empleado
         empName = myReader.GetString(0) + " "
                 + myReader.GetString(1) _
                 + ". " +  myReader.GetString(2)
         empNameNullInd = 0
         salary = myReader.GetDecimal(3)

         If bonus = 0
            If salary > 75000
               bonus = salary * 0.025
               bonusNullInd = 0  ' Devolver un valor distinto de NULL
            Else
               bonus = salary * 0.05
               bonusNullInd = 0  ' Devolver un valor distinto de NULL
            End If
         End If
      Else  ' No se encuentra el empleado
         empName = ""            ' Establecer el parámetro de salida
         empNameNullInd = -1     ' Devolver un valor NULL
      End If

      myReader.Close()
   End If

End Sub
```

Ejemplo 4: Procedimiento en Visual Basic del estilo de parámetros GENERAL que devuelve un conjunto de resultados

Este ejemplo muestra lo siguiente:

- Sentencia CREATE PROCEDURE para un procedimiento en Visual Basic externo que devuelve un conjunto de resultados
- Código Visual Basic para un procedimiento del estilo de parámetros GENERAL que devuelve un conjunto de resultados

Este procedimiento acepta el nombre de una tabla como parámetro. Devuelve un conjunto de resultados que contiene todas las filas de la tabla especificada por el parámetro de entrada. Esto se realiza dejando abierto un `DB2DataReader` para un conjunto de resultados de consulta determinado cuando el procedimiento efectúa la devolución. Específicamente, si no se ejecuta `reader.Close()`, se devolverá el conjunto de resultados.

Tabla 27. Código para crear un procedimiento en Visual Basic que devuelve un conjunto de resultados

```
CREATE PROCEDURE ReturnResultSet(IN tableName VARCHAR(20))
SPECIFIC ReturnResultSet
DYNAMIC RESULT SETS 1
LANGUAGE CLR
PARAMETER STYLE GENERAL
FENCED
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenVbProc.dll:bizLogic.empOps!ReturnResultSet'
```

```
Public Shared Sub ReturnResultSet(byVal tableName As String)

     Dim myCommand As DB2Command
     Dim myReader As DB2DataReader

     myCommand = DB2Context.GetCommand()

     ' Establecer la sentencia de SQL a ejecutar y ejecutarla.
     myCommand.CommandText = "SELECT * FROM " + tableName
     myReader = myCommand.ExecuteReader()

     ' El DB2DataReader contiene el resultado de la consulta.
     ' Este conjunto de resultados se puede devolver con el
     ' procedimiento simplemente NO cerrando el DB2DataReader.
     ' Específicamente, NO ejecutar reader.Close()

   End Sub
```

Ejemplo 5: Procedimiento en Visual Basic del estilo de parámetros SQL que accede a la estructura dbinfo

Este ejemplo muestra lo siguiente:

- Sentencia CREATE PROCEDURE para un procedimiento que accede a la estructura `dbinfo`
- Código Visual Basic para un procedimiento del estilo de parámetros SQL que accede a la estructura `dbinfo`

Para acceder a la estructura `dbinfo`, se debe especificar la cláusula DBINFO en la sentencia CREATE PROCEDURE. No es necesario ningún parámetro para la estructura `dbinfo` en la sentencia CREATE PROCEDURE; no obstante, se debe crear un parámetro para la misma en el código externo de la rutina. Este procedimiento sólo devuelve el valor del nombre de base de datos actual desde el campo `dbname` de la estructura `dbinfo`.

Tabla 28. Código para crear un procedimiento en Visual Basic que accede a la estructura dbinfo

```
CREATE PROCEDURE ReturnDbName(OUT dbName VARCHAR(20))
SPECIFIC ReturnDbName
LANGUAGE CLR
PARAMETER STYLE SQL
DBINFO
FENCED
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenVbProc.dll:bizLogic.empOps!ReturnDbName'
```

```
  Public Shared Sub ReturnDbName(byRef dbName As String, _
                                 byRef dbNameNullInd As Int16, _
                                 byRef sqlState As String, _
                                 byVal funcName As String, _
                                 byVal specName As String, _
                                 byRef sqlMessageText As String, _
                                 byVal dbinfo As sqludf_dbinfo)

    ' Recuperar el nombre de base de datos actual desde la
    ' estructura dbinfo y devolverlo.
    dbName = dbinfo.dbname
    dbNameNullInd = 0  ' Devolver un valor no nulo

    ' Si desea devolver un error definido por el usuario en la
    ' SQLCA puede especificar un SQLSTATE de 5 dígitos definido
    ' por el usuario y el texto de la serie de un mensaje de error.
    ' Por ejemplo:
    '
    ' sqlState = "ABCDE"
    ' msg_token = "Se ha producido un error definido por el usuario"
    '
    ' DB2 devolverá estos valores en la SQLCA. Aparecerá
    ' con el formato de un error sqlState normal de
    ' DB2.
  End Sub
```

Ejemplo 6: Procedimiento en Visual Basic con el estilo PROGRAM TYPE MAIN

Este ejemplo muestra lo siguiente:

- Sentencia CREATE PROCEDURE para un procedimiento que utiliza un estilo de programa principal
- Código Visual Basic para el estilo de parámetros GENERAL WITH NULLS en la utilización de un estilo de programa MAIN

Para implementar una rutina en un estilo de programa principal, se debe especificar la cláusula PROGRAM TYPE en la sentencia CREATE PROCEDURE con el valor MAIN. Se especifican parámetros en la sentencia CREATE PROCEDURE; no obstante, en la implementación del código, los parámetros se pasan a la rutina en un parámetro entero `argc` y una matriz de parámetros `argv`.

Tabla 29. Código para crear un procedimiento en Visual Basic del estilo PROGRAM TYPE MAIN

```
CREATE PROCEDURE MainStyle(IN empId CHAR(6),
                           INOUT bonus Decimal(9,2),
                           OUT empName VARCHAR(60))
SPECIFIC mainStyle
DYNAMIC RESULT SETS 0
LANGUAGE CLR
PARAMETER STYLE GENERAL WITH NULLS
FENCED
PROGRAM TYPE MAIN
EXTERNAL NAME 'gwenVbProc.dll:bizLogic.empOps!Main'
```

Tabla 29. Código para crear un procedimiento en Visual Basic del estilo PROGRAM TYPE MAIN (continuación)

```
Public Shared Sub Main( byVal argc As Int32, _
                        byVal argv As Object())

   Dim myCommand As DB2Command
   Dim myReader As DB2DataReader
   Dim empId As String
   Dim bonus As Decimal
   Dim salary As Decimal
   Dim nullInds As Int16()

   empId = argv(0)  ' argv[0] (IN)    nullInd = argv[3]
   bonus = argv(1)  ' argv[1] (INOUT) nullInd = argv[4]
                    ' argv[2] (OUT)   nullInd = argv[5]
   salary = 0
   nullInds = argv(3)

   If nullInds(0) = -1     ' Comprobar si la entrada empId es nula
      nullInds(1) = -1     ' Devolver un valor de bonificación NULL
      argv(1) = ""         ' Establecer el parámetro de salida empName
      nullInds(2) = -1     ' Devolver un valor empName NULL
      Return
   Else
      ' Si el empleado existe y la bonificación actual es 0,
      ' calcular una nueva bonificación basándose en el salario
      ' del empleado. Devolver nombre y nueva bonificación de éste
      myCommand = DB2Context.GetCommand()
      myCommand.CommandText = _
                "SELECT FIRSTNME, MIDINIT, LASTNAME, SALARY " _
              + " FROM EMPLOYEE " _
              + " WHERE EMPNO = '"+ empId + "'"

      myReader = myCommand.ExecuteReader()


      If myReader.Read()  ' Si se encuentra el registro de empleado
         ' Obtener nombre y apellidos y salario del empleado
         argv(2) = myReader.GetString(0) + " "  _
                 + myReader.GetString(1) + ". " _
             + myReader.GetString(2)
         nullInds(2) = 0
         salary = myReader.GetDecimal(3)

         If bonus = 0
            If salary > 75000
               argv(1) = salary * 0.025
               nullInds(1) = 0  ' Devolver un valor distinto de NULL
            Else
               argv(1) = Salary * 0.05
               nullInds(1) = 0  ' Devolver un valor distinto de NULL
            End If
         End If
      Else  ' No se encuentra el empleado
         argv(2) = ""      ' Establecer el parámetro de salida
         nullInds(2) = -1  ' Devolver un valor NULL
      End If

      myReader.Close()
   End If

End Sub
```

Ejemplo: Soporte de XML y XQuery en el procedimiento de C# .NET CLR

Una vez comprendidos los conceptos básicos de los procedimientos y los fundamentos de las rutinas (CLR) Common Language Runtime .NET, XQuery y XML puede empezar a utilizar procedimientos CLR con sus características XML.

El ejemplo que hay a continuación muestra un procedimiento de C# .NET CLR con parámetros del tipo XML así como el modo de actualizar y consultar datos XML.

Requisitos previos

Antes de trabajar con el ejemplo de procedimientos CLR, puede ser conveniente que lea los temas sobre los conceptos siguientes:

- Rutinas CLR (Common Language Runtime) .NET
- Creación de rutinas CLR .NET desde la ventana de mandatos de DB2
- Beneficios del uso de rutinas

Los ejemplos que hay a continuación utilizan una tabla denominada `xmlDataTable` que se define del siguiente modo:

```
CREATE TABLE xmlDataTable
(
   num INTEGER,
   xdata XML
)

INSERT INTO xmlDataTable VALUES
   (1, XMLPARSE(DOCUMENT '<doc>
                          <type>car</type>
                          <make>Pontiac</make>
                          <model>Sunfire</model>
                          </doc>' PRESERVE WHITESPACE)),
   (2, XMLPARSE(DOCUMENT '<doc>
                          <type>car</type>
                          <make>Mazda</make>
                          <model>Miata</model>
                          </doc>' PRESERVE WHITESPACE)),
   (3, XMLPARSE(DOCUMENT '<doc>
                          <type>person</type>
                          <name>Mary</name>
                          <town>Vancouver</town>
                          <street>Waterside</street>
                          </doc>' PRESERVE WHITESPACE)),
   (4, XMLPARSE(DOCUMENT '<doc>
                          <type>person</type>
                          <name>Mark</name>
                          <town>Edmonton</town>
                          <street>Oak</street>
                          </doc>' PRESERVE WHITESPACE)),
   (5, XMLPARSE(DOCUMENT '<doc>
                          <type>animal</type>
                          <name>dog</name>
                          </doc>' PRESERVE WHITESPACE)),
   (6, NULL),
   (7, XMLPARSE(DOCUMENT '<doc>
                          <type>car</type>
                          <make>Ford</make>
                          <model>Taurus</model>
                          </doc>' PRESERVE WHITESPACE)),
   (8, XMLPARSE(DOCUMENT '<doc>
                          <type>person</type>
                          <name>Kim</name>
                          <town>Toronto</town>
                          <street>Elm</street>
                          </doc>' PRESERVE WHITESPACE)),
```

```
(9, XMLPARSE(DOCUMENT '<doc>
                              <type>person</type>
                              <name>Bob</name>
                              <town>Toronto</town>
                              <street>Oak</street>
                              </doc>' PRESERVE WHITESPACE)),
(10, XMLPARSE(DOCUMENT '<doc>
                               <type>animal</type>
                               <name>bird</name>
                               </doc>' PRESERVE WHITESPACE))@
```

Procedimiento

Utilice los ejemplos siguientes como referencias al crear sus propios procedimientos CLR en C#:

- Archivo de código externo C#
- Ejemplo 1: Procedimiento en C# de estilo de parámetro GENERAL con características XML

Archivo de código externo C#

El ejemplo se compone de dos partes: la sentencia CREATE PROCEDURE y la implementación en código C# externo del procedimiento desde el cual se puede crear el conjunto asociado.

El archivo fuente en C# que contiene las implementaciones de procedimientos de los ejemplos siguientes se denomina `gwenProc.cs` y tiene el formato siguiente:

Tabla 30. Formato del archivo de código externo C#

```
using System;
using System.IO;
using System.Data;
using IBM.Data.DB2;
using IBM.Data.DB2Types;

namespace bizLogic
{
   class empOps
   {          ...
     // C# procedures
              ...
   }
}
```

Las inclusiones del archivo se indican al principio del mismo. La inclusión `IBM.Data.DB2` es necesaria si alguno de los procedimientos del archivo contiene SQL. La inclusión de `IBM.Data.DB2Types` es necesaria si se va a utilizar alguno de los procedimientos del archivo que contenga parámetros o variables del tipo XML. Existe una declaración de espacio de nombres en este archivo y una clase `empOps` que contiene los procedimientos. El uso de espacios de nombres es opcional. Si se utiliza un espacio de nombres, éste debe aparecer en el nombre de vía de acceso de ensamblaje proporcionado en la cláusula EXTERNAL de la sentencia CREATE PROCEDURE.

Es importante tener en cuenta el nombre del archivo, el espacio de nombres y el nombre de la clase, que contiene una implementación de procedimiento determinada. Estos nombres son importantes, ya que la cláusula EXTERNAL de la sentencia CREATE PROCEDURE correspondiente a cada procedimiento debe especificar esta información a fin de que DB2 pueda localizar el conjunto y la clase del procedimiento CLR.

Ejemplo 1: Procedimiento en C# de estilo de parámetro GENERAL con características XML

Este ejemplo muestra lo siguiente:

- Sentencia CREATE PROCEDURE para un procedimiento de estilo de parámetros GENERAL
- Código C# para un procedimiento del estilo de parámetro GENERAL con parámetros XML

Este procedimiento toma dos parámetros, un entero `inNum` y `inXML`. Estos valores se insertan en la tabla `xmlDataTable`. A continuación, se recupera un valor XML utilizando XQuery. Otro valor se recupera utilizando SQL. Los valores XML recuperados se asignan a dos parámetros de salida, `outXML1` y `outXML2`. No se devuelven conjuntos de resultados.

Tabla 31. Código para crear un procedimiento GENERAL tipo parámetro C#

```
CREATE PROCEDURE  xmlProc1   ( IN inNUM  INTEGER,
                               IN inXML  XML as CLOB (1K),
                               OUT inXML  XML as CLOB (1K),
                               OUT inXML  XML as CLOB (1K)
                             )
LANGUAGE CLR
PARAMETER STYLE GENERAL
DYNAMIC RESULT SETS 0
FENCED
THREADSAFE
DETERMINISTIC
NO DBINFO
MODIFIES SQL DATA
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenProc.dll:bizLogic.empOps!xmlProc1' ;
```

```
//*************************************************************************
//  Stored Procedure: xmlProc1
//
//  Purpose:   insert XML data into XML column
//
//  Parameters:
//
//   IN:      inNum -- the sequence of XML data to be insert in xmldata table
//            inXML -- XML data to be inserted
//   OUT:   outXML1 -- XML data returned - value retrieved using XQuery
//          outXML2 -- XML data returned - value retrieved using SQL
//*************************************************************************
```

Tabla 31. Código para crear un procedimiento GENERAL tipo parámetro C# (continuación)

```
public static void xmlProc1 (     int      inNum, DB2Xml   inXML,
                                  out DB2Xml   outXML1, out DB2Xml   outXML2 )
{
   // Create new command object from connection context
   DB2Parameter parm;
   DB2Command cmd;
   DB2DataReader reader = null;
   outXML1 = DB2Xml.Null;
   outXML2 = DB2Xml.Null;

   // Insert input XML parameter value into a table
   cmd = DB2Context.GetCommand();
   cmd.CommandText = "INSERT INTO "
                   + "xmlDataTable( num , xdata ) "
                   + "VALUES( ?, ? )";

   parm = cmd.Parameters.Add("@num", DB2Type.Integer );
   parm.Direction = ParameterDirection.Input;
   cmd.Parameters["@num"].Value = inNum;
   parm = cmd.Parameters.Add("@data", DB2Type.Xml);
   parm.Direction = ParameterDirection.Input;
   cmd.Parameters["@data"].Value = inXML ;
   cmd.ExecuteNonQuery();
   cmd.Close();

   // Retrieve XML value using XQuery
                 and assign value to an XML output parameter
   cmd = DB2Context.GetCommand();
   cmd.CommandText = "XQUERY for $x " +
                   "in db2-fn:xmlcolumn(\"xmlDataTable.xdata\")/doc "+
                   "where $x/make = \'Mazda\' " +
                   "return <carInfo>{$x/make}{$x/model}</carInfo>";
   reader = cmd.ExecuteReader();
   reader.CacheData= true;

   if (reader.Read())
   {  outXML1 = reader.GetDB2Xml(0);  }
   else
   {  outXML1 = DB2Xml.Null;  }

   reader.Close();
   cmd.Close();

   // Retrieve XML value using SQL
                 and assign value to an XML output parameter value
   cmd = DB2Context.GetCommand();
   cmd.CommandText = "SELECT xdata "
                    + "FROM xmlDataTable "
                    + "WHERE num = ?";

   parm = cmd.Parameters.Add("@num", DB2Type.Integer );
   parm.Direction = ParameterDirection.Input;
   cmd.Parameters["@num"].Value = inNum;
   reader = cmd.ExecuteReader();
   reader.CacheData= true;

   if (reader.Read())
   {  outXML2 = reader.GetDB2Xml(0);  }
   else
   {  outXML = DB2Xml.Null;  }

   reader.Close() ;
   cmd.Close();

   return;
}
```

Ejemplo: Soporte de XML y XQuery en el procedimiento de C

Una vez comprendidos los conceptos básicos de los procedimientos y los fundamentos de las rutinas de C, XQuery y XML, podrá empezar a crear y utilizar procedimientos C con sus características XML.

El ejemplo que hay a continuación muestra un procedimiento de C con parámetros del tipo XML así como el modo de actualizar y consultar datos XML.

Requisitos previos

Antes de trabajar con el ejemplo de procedimiento de C, puede ser conveniente que lea el tema sobre los conceptos siguientes:

- Beneficios del uso de rutinas

Los ejemplos que hay a continuación utilizan una tabla denominada `xmlDataTable` que se define del siguiente modo:

```
CREATE TABLE xmlDataTable
(
   num INTEGER,
   xdata XML
)

INSERT INTO xmlDataTable VALUES
   (1, XMLPARSE(DOCUMENT '<doc>
                              <type>car</type>
                              <make>Pontiac</make>
                              <model>Sunfire</model>
                              </doc>' PRESERVE WHITESPACE)),
   (2, XMLPARSE(DOCUMENT '<doc>
                              <type>car</type>
                              <make>Mazda</make>
                              <model>Miata</model>
                              </doc>' PRESERVE WHITESPACE)),
   (3, XMLPARSE(DOCUMENT '<doc>
                              <type>person</type>
                              <name>Mary</name>
                              <town>Vancouver</town>
                              <street>Waterside</street>
                              </doc>' PRESERVE WHITESPACE)),
   (4, XMLPARSE(DOCUMENT '<doc>
                              <type>person</type>
                              <name>Mark</name>
                              <town>Edmonton</town>
                              <street>Oak</street>
                              </doc>' PRESERVE WHITESPACE)),
   (5, XMLPARSE(DOCUMENT '<doc>
                              <type>animal</type>
                              <name>dog</name>
                              </doc>' PRESERVE WHITESPACE)),
   (6, NULL),
   (7, XMLPARSE(DOCUMENT '<doc>
                              <type>car</type>
                              <make>Ford</make>
                              <model>Taurus</model>
                              </doc>' PRESERVE WHITESPACE)),
   (8, XMLPARSE(DOCUMENT '<doc>
                              <type>person</type>
                              <name>Kim</name>
                              <town>Toronto</town>
                              <street>Elm</street>
                              </doc>' PRESERVE WHITESPACE)),
   (9, XMLPARSE(DOCUMENT '<doc>
                              <type>person</type>
                              <name>Bob</name>
                              <town>Toronto</town>
                              <street>Oak</street>
                              </doc>' PRESERVE WHITESPACE)),
   (10, XMLPARSE(DOCUMENT '<doc>
                               <type>animal</type>
                               <name>bird</name>
                               </doc>' PRESERVE WHITESPACE))
```

Procedimiento

Utilice los ejemplos siguientes como referencias al crear sus propios procedimientos C:

- Archivo de código externo C
- Ejemplo 1: Procedimiento en C en estilo de parámetros SQL con características XML

El archivo de código externo C

El ejemplo se compone de dos partes: la sentencia CREATE PROCEDURE y la implementación en código C externo del procedimiento desde el cual se puede crear el conjunto asociado.

El archivo fuente en C que contiene las implementaciones de procedimientos de los ejemplos siguientes se denomina `gwenProc.SQC` y tiene el formato siguiente:

Tabla 32. Formato del archivo de código externo C

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sqlda.h>
#include <sqlca.h>
#include <sqludf.h>
#include <sql.h>
#include <memory.h>

// procedimientos de C
               ...
```

Las inclusiones del archivo se indican al principio del mismo. No se necesitan archivos de incluir adicionales para el soporte de XML en rutinas SQL incorporadas.

Es importante tener en cuenta el nombre del archivo y el nombre de la función que se corresponde con la implementación de procedimiento. Estos nombres son importantes, ya que la cláusula EXTERNAL de la sentencia CREATE PROCEDURE para cada procedimiento debe especificar esta información a fin de que el gestor de base de datos DB2 pueda localizar la biblioteca y el punto de entrada que se corresponden con el procedimiento C.

Ejemplo 1: Procedimiento en C de estilo de parámetros SQL con características XML

Este ejemplo muestra lo siguiente:

- Sentencia CREATE PROCEDURE para un procedimiento del estilo de parámetros SQL
- Código C para un procedimiento del estilo de parámetros SQL con parámetros XML

Este procedimiento recibe dos parámetros de entrada. El primer parámetro de entrada se denomina `inNum` y es del tipo INTEGER. El segundo parámetro de entrada se denomina `inXML` y es del tipo XML. Los valores de los parámetros de entrada se utilizan para insertar una fila en la tabla `xmlDataTable`. A continuación, se recupera un valor XML utilizando una sentencia SQL. Otro valor XML se

recupera utilizando una expresión XQuery. Los valores XML recuperados se asignan respectivamente a dos parámetros de salida, `out1XML` y `out2XML`. No se devuelven conjuntos de resultados.

Tabla 33. Código para crear un procedimiento SQL tipo parámetro C

```
CREATE PROCEDURE  xmlProc1   ( IN inNUM  INTEGER,
                             IN inXML  XML as CLOB (1K),
                             OUT inXML  XML as CLOB (1K),
                             OUT inXML  XML as CLOB (1K)
                           )
LANGUAGE C
PARAMETER STYLE SQL
DYNAMIC RESULT SETS 0
FENCED
THREADSAFE
DETERMINISTIC
NO DBINFO
MODIFIES SQL DATA
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenProc!xmlProc1' ;
```

```
//*************************************************************************
//  Stored Procedure: xmlProc1
//
//  Purpose:   insert XML data into XML column
//
//  Parameters:
//
//   IN:      inNum -- the sequence of XML data to be insert in xmldata table
//            inXML -- XML data to be inserted
//   OUT:   out1XML -- XML data returned - value retrieved using XQuery
//          out2XML -- XML data returned - value retrieved using SQL
//*************************************************************************
```

Tabla 33. Código para crear un procedimiento SQL tipo parámetro C (continuación)

```
#ifdef __cplusplus
extern "C"
#endif
SQL_API_RC SQL_API_FN testSecA1(sqlint32* inNum,
                                SQLUDF_CLOB* inXML,
                                SQLUDF_CLOB* out1XML,
                                SQLUDF_CLOB* out2XML,
                                SQLUDF_NULLIND *inNum_ind,
                                SQLUDF_NULLIND *inXML_ind,
                                SQLUDF_NULLIND *out1XML_ind,
                                SQLUDF_NULLIND *out2XML_ind,
                                SQLUDF_TRAIL_ARGS)
{
  char *str;
  FILE *file;

  EXEC SQL INCLUDE SQLCA;

  EXEC SQL BEGIN DECLARE SECTION;
       sqlint32 hvNum1;
       SQL TYPE IS XML AS CLOB(200) hvXML1;
       SQL TYPE IS XML AS CLOB(200) hvXML2;
       SQL TYPE IS XML AS CLOB(200) hvXML3;
  EXEC SQL END DECLARE SECTION;

  /* Comprobar indicadores nulos para parámetros de entrada */
  if ((*inNum_ind < 0) ||  (*inXML_ind < 0)) {
     strcpy(sqludf_sqlstate, "38100");
     strcpy(sqludf_msgtext, "Received null input");
     return 0;
  }

  /* Copiar parámetros de entrada en variables del sistema principal */
  hvNum1 = *inNum;
  hvXML1.length = inXML->length;
  strncpy(hvXML1.data, inXML->data, inXML->length);

  /* Ejecutar sentencia SQL */
  EXEC SQL
     INSERT INTO xmlDataTable (num, xdata) VALUES (:hvNum1, :hvXML1);

  /* Ejecutar sentencia SQL */
  EXEC SQL
     SELECT xdata INTO :hvXML2
        FROM xmlDataTable
           WHERE num = :hvNum1;

  sprintf(stmt5, "SELECT XMLQUERY('for $x in $xmldata/doc
                                   return <carInfo>{$x/model}</carInfo>'
                                   passing by ref xmlDataTable.xdata
                                       as \"xmldata\" returning sequence)
                    FROM xmlDataTable WHERE num = ?");

  EXEC SQL PREPARE selstmt5 FROM :stmt5 ;
  EXEC SQL DECLARE c5 CURSOR FOR selstmt5;
  EXEC SQL OPEN c5 using :hvNum1;
  EXEC SQL FETCH c5 INTO :hvXML3;

  exit:

  /* Establecer código de retorno de salida */
  *outReturnCode = sqlca.sqlcode;
  *outReturnCode_ind = 0;

  return 0;
}
```

Ejemplos de funciones CLR de .NET en C#

Una vez comprendidos los conceptos básicos de las funciones definidas por el usuario (UDF) y los fundamentos de las rutinas CLR, puede empezar a explotar las

UDF CLR en sus aplicaciones y en el entorno de bases de datos. Este tema contiene algunos ejemplos de UDF CLR para poder empezar. Si desea obtener ejemplos de procedimientos CLR en C#:

- "Ejemplos de procedimientos CLR de .NET en C#" en la página 176

Antes de trabajar con los ejemplos de UDF CLR, puede ser conveniente que lea los temas sobre los conceptos siguientes:

- "Rutinas CLR (Common Language Runtime) .NET" en la página 153
- "Creación de rutinas CLR .NET desde la ventana de mandatos de DB2" en la página 165
- "Funciones escalares externas" en la página 139
- "Creación de código de rutinas .NET CLR" en la página 167

Los ejemplos siguientes utilizan una tabla denominada `EMPLOYEE` que está incluida en la base de datos `SAMPLE`.

Utilice los ejemplos siguientes como referencias al crear sus propias UDF CLR en C#:

- Archivo de código externo C#
- Ejemplo 1: Función de tabla en C# del estilo de parámetro SQL
- Ejemplo 2: Función escalar en C# del estilo de parámetro SQL

Archivo de código externo C#

Los ejemplos siguientes muestran una variedad de implementaciones de UDF en C#. La sentencia CREATE FUNCTION se proporciona para cada UDF con el código fuente C# correspondiente desde el cual se puede crear el conjunto asociado. El archivo fuente en C# que contiene las declaraciones de funciones utilizadas en los ejemplos siguientes se denomina `gwenUDF.cs` y tiene el formato siguiente:

Tabla 34. Formato del archivo de código externo C#

```
using System;
using System.IO;
using IBM.Data.DB2;

namespace bizLogic
{
   ...
   // Definiciones de clases que contienen declaraciones de UDF
   // y cualquier definición de clase de soporte
   ...
}
```

Las declaraciones de funciones deben estar incluidas en una clase dentro de un archivo de C#. El uso de espacios de nombres es opcional. Si se utiliza un espacio de nombres, éste debe aparecer en el nombre de vía de acceso de conjunto proporcionado en la cláusula EXTERNAL de la sentencia CREATE PROCEDURE. La inclusión de `IBM.Data.DB2.` es necesaria si la función contiene SQL.

Ejemplo 1: Función de tabla en C# del estilo de parámetro SQL

Este ejemplo muestra lo siguiente:

- Sentencia CREATE FUNCTION para una función de tabla del estilo de parámetro SQL
- Código C# para una función de tabla del estilo de parámetro SQL

Esta función de tabla devuelve una tabla que contiene filas de los datos de empleado que se han creado a partir de una matriz de datos. Existen dos clases asociadas con este ejemplo. La clase `person` representa los empleados y la clase `empOps` contiene la UDF de tabla de rutina que utiliza la clase `person`. La información sobre el salario de los empleados se actualiza basándose en el valor de un parámetro de entrada. La matriz de datos de este ejemplo se crea dentro de la propia función de tabla en la primera llamada de la función de tabla. Dicha matriz también se podría haber creado leyendo datos de un archivo de texto del sistema de archivos. Los valores de datos de la matriz se graban en un área reutilizable para que sea posible acceder a los datos en llamadas subsiguientes de la función de tabla.

En cada llamada de la función de tabla, se lee un registro de la matriz y se genera una fila en la tabla devuelta por la función. La fila se genera en la tabla estableciendo los parámetros de salida de la función de tabla en los valores de fila deseados. Después de que se produzca la llamada final de la función de tabla, se devuelve la tabla de las filas generadas.

Tabla 35. Código para crear una función de tabla en C# del estilo de parámetro SQL

```
CREATE FUNCTION tableUDF(double)
RETURNS TABLE (name varchar(20),
               job varchar(20),
               salary double)
EXTERNAL NAME 'gwenUDF.dll:bizLogic.empOps!tableUDF'
LANGUAGE CLR
PARAMETER STYLE SQL
NOT DETERMINISTIC
FENCED
THREADSAFE
SCRATCHPAD 10
FINAL CALL
EXECUTION CONTROL SAFE
DISALLOW PARALLEL
NO DBINFO
```

Tabla 35. Código para crear una función de tabla en C# del estilo de parámetro SQL (continuación)

```
// La clase Person es una clase de soporte para
// la función de tabla UDF, tableUDF, siguiente.
class Person
{
     private String name;
     private String position;
     private Int32 salary;

     public Person(String newName, String newPosition, Int32
     newSalary)
     {
        this.name = newName;
        this.position = newPosition;
        this.salary = newSalary;
     }

     public String getName()
     {
        return this.name;
     }

     public String getPosition()
     {
        return this.position;
     }

     public Int32 getSalary()
     {
        return this.salary;
     }
}
```

Tabla 35. Código para crear una función de tabla en C# del estilo de parámetro SQL (continuación)

```
class empOps
{

   public static void TableUDF( Double factor, out String name,
                     out String position, out Double salary,
                     Int16 factorNullInd, out Int16 nameNullInd,
                     out Int16 positionNullInd, out Int16 salaryNullInd,
                     ref String sqlState, String funcName,
                     String specName, ref String sqlMessageText,
                     Byte[] scratchPad, Int32 callType)
   {

      Int16 intRow = 0;

      // Crear una matriz de información del tipo Person
      Person[] Staff = new
      Person[3];
      Staff[0] = new Person("Gwen", "Developer", 10000);
      Staff[1] = new Person("Andrew", "Developer", 20000);
      Staff[2] = new Person("Liu", "Team Leader", 30000);

      salary = 0;
      name = position = "";
      nameNullInd = positionNullInd = salaryNullInd = -1;

      switch(callType)
      {
         case (-2):  // Case SQLUDF_TF_FIRST:
           break;

         case (-1):  // Case SQLUDF_TF_OPEN:
           intRow = 1;
           scratchPad[0] = (Byte)intRow;  // Grabar en área reutilizable
           break;
         case (0):   // Case SQLUDF_TF_FETCH:
           intRow = (Int16)scratchPad[0];
           if (intRow > Staff.Length)
           {
              sqlState = "02000";  // Devolver un error SQLSTATE
           }
           else
           {
              // Generar una fila en la tabla de salida
              // basada en los datos de la matriz Staff.
              name =
              Staff[intRow-1].getName();
              position = Staff[intRow-1].getPosition();
              salary = (Staff[intRow-1].getSalary()) * factor;
              nameNullInd = 0;
              positionNullInd = 0;
              salaryNullInd = 0;
           }
           intRow++;
           scratchPad[0] = (Byte)intRow;  // Grabar área reutilizable
           break;

         case (1):   // Case SQLUDF_TF_CLOSE:
           break;

         case (2):   // Case SQLUDF_TF_FINAL:
           break;
      }
   }
}
```

Ejemplo 2: Función escalar en C# del estilo de parámetro SQL

Este ejemplo muestra lo siguiente:

- Sentencia CREATE FUNCTION para una función escalar del estilo de parámetro SQL
- Código C# para una función escalar del estilo de parámetro SQL

Esta función escalar devuelve un solo valor de cuenta para cada valor de entrada sobre el que actúa. Para un valor de entrada situado en la n[a] posición del conjunto de valores de entrada, el valor escalar de salida es el valor n. En cada llamada de la función escalar, donde una llamada está asociada con cada fila o valor del conjunto de filas o valores de entrada, la cuenta aumenta en uno y se devuelve el valor actual de la cuenta. Luego, la cuenta se guarda en el almacenamiento intermedio de memoria del área reutilizable para mantener el valor de cuenta entre cada llamada de la función escalar.

Esta función escalar se puede invocar fácilmente si, por ejemplo, se dispone de una tabla definida del modo siguiente:

```
CREATE TABLE T (i1 INTEGER);
INSERT INTO T VALUES 12, 45, 16, 99;
```

Se puede utilizar una consulta simple como la siguiente para invocar la función escalar:

```
SELECT countUp(i1) as count, i1 FROM T;
```

La salida de una consulta como la indicada sería:

```
COUNT           I1
-----------     ----------
1               12
2               45
3               16
4               99
```

Esta UDF escalar es bastante simple. En lugar de devolver sólo la cuenta de las filas, puede utilizar una función escalar que formatee los datos de una columna existente. Por ejemplo, puede añadir una serie a cada valor de una columna de direcciones, puede crear una serie compleja a partir de una cadena de series de entrada o puede efectuar un cálculo matemático complejo con un conjunto de datos donde deberá almacenar un resultado intermedio.

Tabla 36. Código para crear una función escalar en C# del estilo de parámetro SQL

```
CREATE FUNCTION countUp(INTEGER)
RETURNS INTEGER
LANGUAGE CLR
PARAMETER STYLE SQL
SCRATCHPAD 10
FINAL CALL
NO SQL
FENCED
THREADSAFE
NOT DETERMINISTIC
EXECUTION CONTROL SAFE
EXTERNAL NAME 'gwenUDF.dll:bizLogic.empOps!CountUp';
```

Tabla 36. Código para crear una función escalar en C# del estilo de parámetro SQL (continuación)

```
class empOps
{
   public static void CountUp(     Int32 input,
                                out Int32 outCounter,
                                    Int16 inputNullInd,
                                out Int16 outCounterNullInd,
                                ref String sqlState,
                                    String funcName,
                                    String specName,
                                ref String sqlMessageText,
                                    Byte[] scratchPad,
                                    Int32 callType)
   {
      Int32 counter = 1;

      switch(callType)
      {
         case -1: // case SQLUDF_FIRST_CALL
           scratchPad[0] = (Byte)counter;
           outCounter = counter;
           outCounterNullInd = 0;
           break;
         case 0:  // case SQLUDF_NORMAL_CALL:
           counter = (Int32)scratchPad[0];
           counter = counter + 1;
           outCounter = counter;
           outCounterNullInd = 0;
           scratchPad[0] =
           (Byte)counter;
           break;
         case 1:  // case SQLUDF_FINAL_CALL:
           counter =
           (Int32)scratchPad[0];
           outCounter = counter;
           outCounterNullInd = 0;
           break;
         default: // Nunca se debe entrar aquí
                  // * Necesario para que durante la compilación se
                  //   establezca siempre el parámetro de salida outCounter *
           outCounter = (Int32)(0);
           outCounterNullInd = -1;
           sqlState="ABCDE";
           sqlMessageText = "Should not get here: Default
           case!";
           break;
      }
   }
}
```

Rutinas C y C++

Las rutinas C y C++ son rutinas externas que se crean ejecutando una sentencia CREATE PROCEDURE, CREATE FUNCTION o CREATE METHOD, que haga referencia a una biblioteca creada a partir del código fuente C o C++ como su cuerpo de código externo.

Las rutinas C y C++ pueden ejecutar opcionalmente sentencias de SQL incluyendo sentencias de SQL incorporado.

Los siguientes términos son importantes en el contexto de las rutinas C y C++:

CREATE, sentencia
La sentencia CREATE del lenguaje SQL utilizada para crear la rutina en la base de datos.

Código fuente del cuerpo de la rutina
El archivo de código fuente que contiene la implementación de la rutina C o C++ que corresponde a la especificación de la cláusula EXTERNAL de la sentencia CREATE.

Precompilador
El programa de utilidad de DB2 que preanaliza la implementación del código fuente de la rutina para validar las sentencias de SQL contenidas en el código y genera un paquete.

Compilador
El software específico del lenguaje de programación necesario para compilar y enlazar la implementación del código fuente.

Paquete
El archivo que contiene la información de vía de acceso de tiempo de ejecución que utilizará DB2 en el momento de la ejecución de la rutina para ejecutar las sentencias de SQL contenidas en la implementación del código de la rutina.

Biblioteca de la rutina
Un archivo que contiene el formato compilado del código fuente de la rutina. En Windows a veces se denomina una DLL, porque estos archivos tienen la extensión de archivo .dll.

Antes de desarrollar una rutina C o C++, es importante comprender los conceptos básicos de las rutinas y las características exclusivas y específicas de las rutinas C y C++. También es importante comprender la API de SQL incorporado y los conceptos básicos sobre el desarrollo de aplicaciones de SQL incorporado. Para obtener más información sobre estos temas, consulte los siguientes apartados:

- Rutinas externas
- SQL incorporado
- Archivos de inclusión para rutinas C y C++
- parámetros en rutinas C y C++
- Restricciones de rutinas C y C++

El desarrollo de rutinas C o C++ incluye seguir una serie de instrucciones paso a paso y consultar ejemplos de rutinas C o C++. Consulte:

- Creación de rutinas C y C++
- Ejemplos de procedimientos C
- Ejemplos de funciones definidas por el usuario C

Soporte para el desarrollo de rutinas externas en C

Para desarrollar rutinas externas en C debe utilizar compiladores y software de desarrollo soportados.

Todos los compiladores y el software de desarrollo soportados para el desarrollo de aplicaciones de bases de datos DB2 en C se puede utilizar para el desarrollo de rutinas externas en C.

Soporte para el desarrollo de rutinas externas en C++

Para desarrollar rutinas externas en C++ debe utilizar compiladores y software de desarrollo soportados.

Todos los compiladores y el software de desarrollo soportados para el desarrollo de aplicaciones de bases de datos DB2 en C se pueden utilizar para el desarrollo de rutinas externas en C++.

Herramientas para el desarrollo de rutinas C y C++

Las herramientas soportadas para las rutinas C y C++ son las mismas que las soportadas para aplicaciones C y C++ de SQL incorporado.

No hay entornos de desarrollo de DB2 ni herramientas gráficas de interfaz de usuario para desarrollar, depurar o desplegar aplicaciones o rutinas de SQL incorporado.

Las siguientes interfaces de línea de mandatos se suelen utilizar para desarrollar, depurar y desplegar aplicaciones y rutinas de SQL incorporado:

- Procesador de línea de mandatos de DB2
- Ventana de mandatos de DB2

Estas interfaces dan soporte a la ejecución de las sentencias de SQL necesarias para crear rutinas en una base de datos. El mandato PREPARE y el mandato BIND necesarios para crear rutinas C y C++ que contienen SQL incorporado también se puede emitir desde estas interfaces.

Diseño de rutinas C y C++

El diseño de rutinas C y C++ es una tarea que debe preceder a la creación de dichas rutinas. El diseño de rutinas C y C++ suele estar relacionado con el diseño de rutinas externas implementadas en otros lenguajes de programación y con el diseño de aplicaciones de SQL incorporado.

Requisitos previos

- Conocimientos generales sobre rutinas externas
- Experiencia con la programación de C o C++
- Opcional: Conocimientos sobre, y experiencia con, el desarrollo de aplicaciones CLI y SQL incorporado (si la rutina va a ejecutar sentencias de SQL)

Los temas siguientes proporcionan parte de la información necesaria sobre requisitos previos.

Para obtener más información sobre las funciones y usos de rutinas externas:

- Consulte el tema “Implementación de rutinas externas” en la página 22

Para obtener más información sobre las características de la API de SQL:

- Consulte el tema “Información preliminar sobre el SQL incorporado” en *Developing Embedded SQL Applications*

Con el conocimiento de los requisitos previos, el diseño de rutinas de SQL incorporado consiste principalmente en conocer las funciones y características exclusivas de las rutinas C y C++:

- “Archivo de inclusión necesario para el desarrollo de rutinas C y C++ (sqludf.h)” en la página 219

- “Parámetros en rutinas C y C++” en la página 220
- “Estilo de parámetro de SQL en procedimientos C y C++” en la página 222
- “Estilo de parámetro de SQL en funciones C y C++” en la página 225
- “Manejo de tipos de datos de SQL en rutinas C y C++” en la página 236
- “Variables gráficas del lenguaje principal en rutinas C y C++” en la página 257
- “Devolución de conjuntos de resultados procedentes de procedimientos C y C++” en la página 259
- “Decoración de tipos de C++” en la página 257
- “Restricciones para rutinas externas” en la página 135

Después de conocer las características de C y C++, puede interesarle consultar lo siguiente:

- “Creación de rutinas C y C++” en la página 261

Archivo de inclusión necesario para el desarrollo de rutinas C y C++ (sqludf.h)

El archivo de inclusión `sqludf.h` contiene estructuras, definiciones y valores que son necesarios al codificar implementaciones de rutinas. Aunque este archivo tenga 'udf' en su nombre, (por razones históricas) también resulta útil para procedimientos y métodos almacenados. Al compilar la rutina, debe hacer referencia al directorio `include` de instalación de DB2 que contiene este archivo.

La utilización de objetos en este archivo es recomendable para garantizar que se utilice el tipo de datos C correcto para su sistema operativo en concreto y los sistemas operativos independientes.

El archivo `sqludf.h` contiene definiciones y descripciones de estructura de las definiciones de estructura. A continuación puede ver un breve resumen de su contenido:

- Las definiciones de macros para tipo de datos de SQL que se soportan como parámetros para rutinas externas no precisan representación como una estructura C o C++. En el archivo, las definiciones tienen formatos de nombre como: SQLUDF_x ySQLUDF_x_FBD donde x es un nombre de tipo de datos de SQL y FBD representa FOR BIT DATA para aquellos tipos de datos que se almacenan en formularios binarios.

  También se incluye un tipo de lenguaje C para un argumento o resultado definido con la cláusula AS LOCATOR. Esto sólo es aplicable a las UDF y los métodos.
- Definiciones de estructura C necesarias para representar los siguientes tipos de datos de SQL y parámetros especiales:
  - tipo de datos VARCHAR FOR BIT DATA
  - tipo de datos LONG VARCHAR
  - tipo de datos LONG VARCHAR FOR BIT DATA
  - tipo de datos LONG VARGRAPHIC
  - tipo de datos BLOB
  - tipo de datos CLOB
  - tipo de datos DBCLOB
  - estructura scratchpad (de área reutilizable)
  - estructura dbinfo

Cada una de estas estructuras está representada por una estructura con más de un valor de campo en lugar de por un único tipo de datos C.

L estructura scratchpad define un almacenamiento intermedio que se pasa a una función definida por el usuario para utilizarla durante la invocación de la función. De forma distinta a una variable, los datos almacenados en un área reutilizable son persistentes entre varias llamadas a la función definidas por el usuario dentro de una única invocación. Esto puede resultar útil tanto para las funciones que devuelven valores agregados como para las funciones que necesitan lógica de configuración inicial.

La estructura dbinfo es una estructura que contiene información sobre bases de datos y rutinas, que puede pasarse a la implementación de la rutina, y recibirse de ésta, como un argumento extra, sólo si la cláusula DBINFO está incluida en la sentencia CREATE para dicha rutina.

- Definición de tipos de lenguaje C para argumentos del área reutilizable y tipo de llamada. Se especifica una definición del tipo enum para el argumento tipo de llamada.

  Las funciones externas definidas por el usuario se invocan varias veces para un conjunto de valores. Los tipos de llamada se utilizan para identificar invocaciones de funciones externas individuales. Cada invocación se identifica con un valor de tipo de llamada al que se puede hacer referencia dentro de la lógica de la función. Por ejemplo, hay tipos de llamada especiales para la primera invocación de una función, para las llamadas de captación de datos y para la invocación final. Los tipos de llamada resultan útiles porque se puede asociar lógica específica a un tipo de llamada determinado. Los ejemplos de tipos de llamada incluyen: llamada FIRST, llamada FETCH y llamada FINAL.
- Se definen macros para definir los argumentos de cola necesarios en prototipos de funciones definidas por el usuario (UDF). Los argumentos de cola incluyen los argumentos de invocación UDF estado_SQL, nombre_función, nombre_específico, mensaje_diagnóstico, área_reutilizable y tipo_llamada. También se incluyen las definiciones para hacer referencia a estas construcciones y los diversos valores de SQLSTATE válidos. Se suministran varias definiciones de macro que difieren en su inclusión o exclusión de los argumentos de área reutilizable y tipo de llamada. Éstos corresponden a la presencia o ausencia de la utilización del la cláusula SCRATCHPAD y FINAL CALL en la definición de función.

  Normalmente, cuando se define una función definida por el usuario, es recomendable utilizar la macro SQLUDF_TRAIL_ARGS para simplificar el prototipo de función. A continuación encontrará un ejemplo:

```
void SQL_API_FN ScalarUDF(SQLUDF_CHAR *inJob,
                          SQLUDF_DOUBLE *inSalary,
                          SQLUDF_DOUBLE *outNewSalary,
                          SQLUDF_SMALLINT *jobNullInd,
                          SQLUDF_SMALLINT *salaryNullInd,
                          SQLUDF_SMALLINT *newSalaryNullInd,
                          SQLUDF_TRAIL_ARGS)
```

- Definiciones de macro que se pueden utilizar para probar si argumentos SQL tienen valores vacíos.

Para ver cómo se utilizan distintas definiciones, macros y estructuras que están definidas en el archivo `sqludf.h`, vea las aplicaciones y rutinas de muestra de C y C++.

Parámetros en rutinas C y C++

La declaración de parámetros de las rutinas C y C++ debe cumplir con los requisitos de uno de los estilos de parámetros y del tipo de programa que se

soportan. Si la rutina ha de emplear un área reutilizable o la estructura dbinfo o ha de tener la interfaz de parámetros PROGRAM TYPE MAIN, existen detalles adicionales a considerar, incluidos:

- "Estilos de parámetros soportados para rutinas C y C++"
- "Indicadores nulos de parámetros en rutinas C y C++"
- "Estilo de parámetro de SQL en procedimientos C y C++" en la página 222
- "Estilo de parámetro de SQL en funciones C y C++" en la página 225
- "Paso de parámetros por valor o por referencia en rutinas C y C++" en la página 227
- "No se necesitan parámetros para conjuntos de resultados de procedimientos C y C++" en la página 228
- "Estructura dbinfo como parámetro de rutina C o C++" en la página 228
- "Área reutilizable como parámetro de función C o C++" en la página 230
- "Soporte de MAIN de tipo de programa para procedimientos C y C++" en la página 231

Es muy importante implementar la interfaz de parámetros para rutinas C y C++ correctamente. Esto puede realizarse fácilmente teniendo cuidado de asegurarse de que se eligen e implementan el estilo de parámetro y los tipos de datos correctos de acuerdo con la especificación.

Estilos de parámetros soportados para rutinas C y C++

Los siguientes estilos de parámetros reciben soporte para rutinas C y C++:

- SQL (soportado para procedimientos y funciones; recomendado)
- GENERAL (Soportado para procedimientos)
- GENERAL WITH NULLS (Soportado para procedimientos)

Se recomienda utilizar el estilo de parámetro SQL para todas las rutinas C y C++. Este estilo de parámetro da soporte a valores NULL, proporciona una interfaz estándar para errores de informe, así como áreas reutilizables de soporte y tipos de llamadas.

Para especificar el estilo de parámetro que se van a utilizar para una rutina, debe especificar la cláusula PARAMETER STYLE de la sentencia CREATE para dicha rutina en el momento de creación de la rutina.

El estilo de parámetro se debe reflejar adecuadamente en la implementación del código de la rutina C o C++.

Para obtener más información sobre estos estilos de parámetros, consulte: "Sintaxis para pasar parámetros a rutinas C y C++".

Indicadores nulos de parámetros en rutinas C y C++

Si el estilo de parámetros elegido para una rutina C o C++ (procedimiento o función) necesita que se especifique un parámetro de indicador nulo para cada uno de los parámetros SQL, como en el caso del estilo de parámetros SQL y GENERAL, los indicadores nulos se tienen que pasar como parámetros del tipo de datos SQLUDF_NULLIND*. Para el estilo de parámetros GENERAL WITH NULLS, se deben pasar como una matriz de tipo SQLUDF_NULLIND. Este tipo de datos se define en el archivo de inclusión de una rutina y aplicación de SQL incorporado: sqludf.h.

Los parámetros de indicadores nulos indican si el valor del parámetro correspondiente es equivalente a NULL en SQL o si tiene un valor literal. Si el valor del indicador nulo para un parámetro es 0, significa que el valor del parámetro no es nulo. Si el valor del indicador nulo de un parámetro es -1, se considera que el parámetro tiene un valor equivalente al valor de SQL NULL.

Cuando se utilizan indicadores nulos, es importante incluir código dentro de la rutina que:

- Compruebe los valores de los indicadores nulos para ver los parámetros de entrada antes de utilizarlos.
- Establezca valores de indicadores nulos para parámetros de salida antes de que la rutina devuelva un valor.

Para obtener más información sobre parámetros de SQL, consulte:

- "Estilos de parámetros de rutinas externas" en la página 133
- "Estilo de parámetro de SQL en procedimientos C y C++"
- "Estilo de parámetro de SQL en funciones C y C++" en la página 225

Estilo de parámetro de SQL en procedimientos C y C++

Los procedimientos C y C++ se deben crear utilizando la cláusula PARAMETER STYLE SQL en la sentencia CREATE PROCEDURE. Los convenios de paso de parámetros de este estilo de parámetro se deben implementar en la implantación del código del procedimiento correspondiente.

La implementación de la signatura SQL PARAMETER STYLE de C y C++ necesaria para procedimientos sigue este formato:

```
     SQL_API_RC SQL_API_FN nombre-función ( argumentos-SQL,
ind-argumento-SQL,
                                           sqlstate,
                                           nombre-rutina,
                                           nombre-específico,
                                           mensaje-diagnóstico )
```

SQL_API_RC SQL_API_FN

SQL_API_RC y SQL_API_FN son macros que especifican el tipo de retorno y el convenio de llamada para un procedimiento C o C++, los cuales pueden ser distintos según los sistemas operativos soportados. El uso de estas macros es obligatorio para rutinas C y C++. Las macros se declaran en el archivo de inclusión de rutinas y aplicaciones de SQL sqlsystm.h.

nombre-función

Nombre de la función C o C++ dentro del archivo del código. No es necesario que este valor coincida con el nombre del procedimiento especificado dentro de la sentencia CREATE PROCEDURE correspondiente. Sin embargo, se debe especificar este valor, en combinación con el nombre de la biblioteca, en la cláusula EXTERNAL NAME para identificar el punto de entrada de función correcto dentro de la biblioteca que se va a utilizar. En las rutinas C++, el compilador C++ aplica la decoración de tipos al nombre de punto de entrada. Hay que especificar el nombre decorado con tipos en la cláusula EXTERNAL NAME o bien se debe definir el punto de entrada como extern "C" en el código del usuario. Debe exportar el nombre de la función de forma explícita.

argumentos-SQL

Los argumentos de C o C++ correspondientes al conjunto de parámetros

SQL especificados en la sentencia CREATE PROCEDURE. Los parámetros de modalidad IN, OUT e INOUT se pasan mediante valores de puntero individuales.

ind-argumento-SQL
Indicadores de nulo C o C++ que corresponden al conjunto de parámetros de SQL especificados en la sentencia CREATE PROCEDURE. Para cada parámetro de modalidad IN, OUT e INOUT, debe existir un parámetro del indicador de nulo asociado. Los indicadores de nulo se pueden pasar como argumentos individuales de tipo SQLUDF_NULLIND, o como parte de una sola matriz de indicadores de nulo definida como SQLUDF_NULLIND*.

sqlstate Valor del parámetro de entrada y salida utilizado por la rutina para señalar condiciones de aviso o error. Normalmente, este argumento se utiliza para asignar un valor de SQLSTATE definido por el usuario correspondiente a un error o aviso, que puede devolverse al llamador. Los valores SQLSTATE con formato `38xxx`, donde `xxx` representa cualquier valor numérico, están disponibles para valores de error SQLSTATE definidos por el usuario. Los valores con formato `01Hxx`, donde `xx` representa cualquier valor numérico, están disponibles para valores de avisos SQLSTATE definidos por el usuario.

nombre-rutina
Valor del parámetro de entrada que contiene el nombre calificado de la rutina. Este valor lo genera DB2 y lo pasa a la rutina en el formato `<nombre-esquema>.<nombre-rutina>` donde <nombre-esquema> y <nombre-rutina> corresponden respectivamente al valor de la columna ROUTINESCHEMA y al valor de la columna ROUTINENAME correspondientes a la rutina dentro de la vista de catálogo SYSCAT.ROUTINES. Este valor puede resultar útil si múltiples definiciones de rutina diferentes utilizan una única implementación de rutina. Cuando el nombre de la definición de la rutina se ha pasado a la rutina, se podrá ejecutar condicionalmente la lógica, dependiendo de qué definición se haya utilizado. El nombre de la rutina puede ser útil cuando se formule la información de diagnósticos, incluidos los mensajes de error, o cuando se escriba en un archivo de anotaciones cronológicas.

nombre-específico
Valor del parámetro de entrada que contiene el nombre exclusivo específico de la rutina. Este valor lo genera DB2 y lo pasa a la rutina. Este valor se corresponde con el valor de la columna SPECIFICNAME correspondiente a la rutina en la vista SYSCAT.ROUTINES. Puede resultar tan útil como el nombre-rutina.

mensaje-diagnóstico
Valor del parámetro de salida opcionalmente utilizado por la rutina para devolver el texto del mensaje a la aplicación o rutina que lo ha invocado. Este parámetro está diseñado para utilizarse como complemento para el argumento SQLSTATE. Se puede utilizar para asignar un mensaje de error definido por el usuario que acompañe a un valor SQLSTATE definido por el usuario, lo cual puede proporcionar información más detallada sobre errores o avisos de diagnóstico a quien llama la rutina.

Nota: Para simplificar la escritura de signaturas de procedimientos C y C++, puede utilizarse la definición de macro SQLUDF_TRAIL_ARGS definida en

sqludf.h, en lugar de utilizar argumentos individuales, en la signatura del procedimiento para implementar los argumentos de tipos de datos que no son SQL.

A continuación se muestra un ejemplo de una implementación de un procedimiento C o C++ que acepta un solo parámetro de entrada y devuelve un solo parámetro de salida y un conjunto de resultados:

```
/*****************************************************************
 Rutina:    cstp

  Objetivo:  Devuelve un valor de parámetro de salida basado en un valor
             de parámetro de entrada

            Muestra cómo:
             - definir un procedimiento utilizando PARAMETER STYLE SQL
             - definir indicadores NULL para el parámetro
             - ejecutar una sentencia de SQL
             - definir un indicador NULL cuando el parámetro no es nulo

  Parámetros:

  IN:       inParm
  OUT:      outParm

            Cuando se define PARAMETER STYLE SQL para la rutina
            (consulte el script de registro de rutina spcreate.db2),
            además de los parámetros pasados durante la invocación,
            los argumentos siguientes se pasan a la rutina en el
            orden siguiente:

             - un indicador de nulo para cada parámetro IN/INOUT/OUT,
               ordenados de forma que coincidan con el orden de las
               declaraciones de parámetros
             - SQLSTATE que se debe devolver a DB2 (salida)
             - nombre calificado de la rutina (entrada)
             - nombre específico de la rutina (entrada)
             - serie de diagnóstico de SQL para devolver un texto de
               mensajes de error opcional a DB2 (salida)

            Consulte más abajo las declaraciones de parámetros reales
            para ver qué tipos de datos y tamaños se recomiendan.

            SUGERENCIA DE CÓDIGO:
            --------
            En lugar de codificar los parámetros 'extra':
              sqlstate, nombre calificado de la rutina, nombre
              específico de la rutina, mensaje de diagnóstico,
            se puede utilizar una macro SQLUDF_TRAIL_ARGS.
            Esta macro se define en el archivo de inclusión sqludf.h
            de DB2

            SUGERENCIA DE EJEMPLO:
            ------------
            El método siguiente es equivalente al prototipo real
            que se utiliza y hace uso de las definiciones de macro
            incluidas en sqludf.h.  El formato implementado real
            es más sencillo y elimina los problemas de tipo de
            datos.
extern "C"  SQL_API_RC SQL_API_FN OutLanguage(
                                              sqlint16 *inParm,
                                              double *outParm,
                                              sqlint16 *inParmNullInd,
                                              sqlint16 *outParmNullInd,
                                              char sqlst[6],
                                              char qualName[28],
```

```
                                                  char specName[19],
                                                  char diagMsg[71])
                                                 )

****************************************************************/

extern "C" SQL_API_RC SQL_API_FN cstp ( sqlint16 *inParm,
                                        double *outParm,
                                        SQLUDF_NULLIND *inParmNullInd,
                                        SQLUDF_NULLIND *outParmNullInd,
                                        SQLUDF_TRAIL_ARGS )
{
  EXEC SQL INCLUDE SQLCA;

  EXEC SQL BEGIN DECLARE SECTION;
     sqlint16 sql_inParm;
  EXEC SQL END DECLARE SECTION;

  sql_inParm = *inParm;

  EXEC SQL DECLARE cur1 CURSOR FOR
   SELECT value
   FROM table01
    WHERE index = :sql_inParm;

  *outParm = (*inParm) + 1;
  *outParmNullInd = 0;

  EXEC SQL OPEN cur1;

  return (0);
}
```

La sentencia CREATE PROCEDURE que corresponde a este procedimiento es la siguiente:

```
CREATE PROCEDURE cproc( IN inParm INT, OUT outParm INT )
LANGUAGE c
PARAMETER STYLE sql
DYNAMIC RESULT SETS 1
FENCED
THREADSAFE
RETURNS NULL ON NULL INPUT
EXTERNAL NAME 'c_rtns!cstp'
```

La sentencia anterior supone que la implementación del procedimiento C o C++ está en un archivo de biblioteca denominado `c_rtns` y en una función denominada `cstp`.

Estilo de parámetro de SQL en funciones C y C++

Las funciones definidas por el usuario C y C++ se deben crear utilizando la cláusula PARAMETER STYLE SQL en la sentencia CREATE FUNCTION. Los convenios de paso de parámetros de este estilo de parámetro se deben implementar en la implantación del código fuente correspondiente. La implementación de la signatura SQL PARAMETER STYLE de C y C++ necesaria para funciones definidas por el usuario sigue este formato:

```
SQL_API_RC SQL_API_FN nombre-función ( argumentos-SQL,
 ind-argumento-SQL,
 SQLUDF_TRAIL_ARGS )
```

SQL_API_RC SQL_API_FN

SQL_API_RC y SQL_API_FN son macros que especifican el tipo de retorno y el convenio de llamada para una función definida por el usuario C o C++, los cuales pueden ser distintos según los sistemas operativos soportados. El

uso de estas macros es obligatorio para rutinas C y C++. Las macros se declaran en el archivo de inclusión de rutinas y aplicaciones de SQL incorporado `sqlsystm.h`.

nombre-función
Nombre de la función C o C++ dentro del archivo del código. No es necesario que este valor coincida con el nombre de la función especificado dentro de la sentencia CREATE FUNCTION correspondiente. Sin embargo, se debe especificar este valor, en combinación con el nombre de la biblioteca, en la cláusula EXTERNAL NAME para identificar el punto de entrada de función correcto dentro de la biblioteca que se va a utilizar. En las rutinas C++, el compilador C++ aplica la decoración de tipos al nombre de punto de entrada. Se tiene que especificar el nombre decorado del tipo en la cláusula EXTERNAL NAME o la declaración de la función dentro del archivo del código fuente se tiene que preceder de `extern "C"`, tal como se muestra en el siguiente ejemplo: extern "C" SQL_API_RC SQL_API_FN OutLanguage(char *, sqlint16 *, char *, char *, char *, char *);

argumentos-SQL
Los argumentos de C o C++ correspondientes al conjunto de parámetros SQL especificados en la sentencia CREATE FUNCTION.

ind-argumento-SQL
Para cada argumento-SQL, se necesita un parámetro de indicador nulo para especificar si el valor del parámetro se tiene que interpretar dentro de la implementación de la rutina como un valor NULL en SQL. Los indicadores nulos se deben especificar con el tipo de datos SQLUDF_NULLIND. Este tipo de datos se define en el archivo de inclusión de rutinas de SQL incorporado `sqludf.h`.

SQLUDF_TRAIL_ARGS
Una macro definida en el archivo de inclusión de rutinas de SQL incorporado `sqludf.h` que, una vez ampliada, define los argumentos finales adicionales necesarios para obtener una signatura completa SQL de estilo de parámetro. Se pueden utilizar dos macros: SQLUDF_TRAIL_ARGS y SQLUDF_TRAIL_ARGS_ALL. SQLUDF_TRAIL_ARGS, una vez ampliada, tal como está definida en sqludf.h, equivale a la adición de los siguientes argumentos de rutina:

```
SQLUDF_CHAR *sqlState,
SQLUDF_CHAR  qualName,
SQLUDF_CHAR  specName,
SQLUDF_CHAR *sqlMessageText,
```

En general estos argumentos no son necesarios ni se suelen utilizar como parte de la lógica de función definida por el usuario. Representan el valor de SQLSTATE de salida que se tiene que pasar al que ha invocado la función, el nombre de función calificado al completo de entrada, el nombre específico de la función de entrada y el texto del mensaje de salida que se tiene que devolver con SQLSTATE. SQLUDF_TRAIL_ARGS, una vez ampliada, tal como está definida en sqludf.h, equivale a la adición de los siguientes argumentos de rutina:

```
SQLUDF_CHAR  qualName,
SQLUDF_CHAR  specName,
SQLUDF_CHAR  sqlMessageText,
SQLUDF_SCRAT *scratchpad
SQLUDF_CALLT *callType
```

Si la sentencia UDF CREATE incluye la cláusula SCRATCHPAD o la cláusula FINAL CALL, se debe utilizar la macro `SQLUDF_TRAIL_ARGS_ALL`.

Además de los argumentos proporcionados con SQLUDF_TRAIL_ARGS, esta macro también contiene punteros a una estructura de área reutilizable y a un valor de tipo de llamada.

A continuación se muestra un ejemplo de una UDF C o C++ que devuelve en un parámetro de salida el valor del producto de sus dos valores de parámetros de entrada:

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

  /* comprobar que los valores de los parámetros de entrada no sean nulos
     comprobando los valores de indicadores nulos correspondientes
         0  : indica que el valor del parámetro no es NULL
         -1 : indica que el valor del parámetro es NULL

    Si los valores no son NULL, calcular el producto.
    Si los valores son NULL, devolver un valor de salida NULL. */

  if ((*in1NullInd != -1) &&
       *in2NullInd != -1))
  {
    *outProduct = (*in1) * (*in2);
    *productNullInd = 0;
  }
  else
  {
    *productNullInd = -1;
  }
  return (0);
}
```

La sentencia CREATE FUNCTION correspondiente que se puede utilizar para crear esta UDF podría ser la siguiente:

```
CREATE FUNCTION product( in1 DOUBLE, in2 DOUBLE )
  RETURNS DOUBLE
  LANGUAGE C
  PARAMETER STYLE SQL
  NO SQL
  FENCED THREADSAFE
  DETERMINISTIC
  RETURNS NULL ON NULL INPUT
  NO EXTERNAL ACTION
  EXTERNAL NAME 'c_rtns!product'
```

En la sentencia de SQL anterior se da por supuesto que la función C o C++ está en un archivo de biblioteca en el directorio de función denominado c_rtns.

Paso de parámetros por valor o por referencia en rutinas C y C++

Para rutinas C y C++, los valores de los parámetros se deben pasar siempre por referencia a las rutinas utilizando punteros. Esto es necesario para parámetros sólo de entrada, de entrada-salida y de salida. por referencia.

Los parámetros del indicador de nulo también se deben pasar por referencia a las rutinas utilizando punteros.

Nota: DB2 controla la asignación de memoria para todos los parámetros y mantiene las referencias de C o C++ a todos los parámetros pasados a una rutina o salidos de ella. No es necesario asignar o liberar la memoria asociada a parámetros de rutinas e indicadores de nulo.

No se necesitan parámetros para conjuntos de resultados de procedimientos C y C++

No se necesita ningún parámetro en la signatura de la sentencia CREATE PROCEDURE para un procedimiento o en la implementación de procedimiento asociado para devolver un conjunto de resultados al llamador.

Los conjuntos de resultados devueltos de procedimientos C, se devuelven utilizando cursores.

Si desea más información sobre la devolución de conjuntos de resultados desde procedimientos LANGUAGE C, consulte:

- "Devolución de conjuntos de resultados procedentes de procedimientos C y C++" en la página 259

Estructura dbinfo como parámetro de rutina C o C++

La estructura `dbinfo` es una estructura que contiene información sobre bases de datos y rutinas, que puede pasarse a la implementación de la rutina, y recibirse de ésta, como un argumento extra, sólo si la cláusula DBINFO está incluida en la sentencia CREATE para dicha rutina.

La estructura dbinfo recibe soporte en rutinas LANGUAGE C a través del uso de la estructura `sqludf_dbinfo`. Esta estructura C está definida en el archivo de inclusión de DB2, `sqludf.h` situado en el directorio `sqllib\include`.

La estructura `sqludf_dbinfo` se define del modo siguiente:

```
SQL_STRUCTURE sqludf_dbinfo
{
   unsigned short  dbnamelen;                      /* Long. del nombre de base datos */
   unsigned char   dbname[SQLUDF_MAX_IDENT_LEN];   /* Nombre de la base de datos     */
   unsigned short  authidlen;                      /* Longitud del ID de autorización*/
   unsigned char   authid[SQLUDF_MAX_IDENT_LEN];   /* ID de autorización             */
   union db_cdpg   codepg;                         /* Página de códigos de base datos*/
   unsigned short  tbschemalen;                    /* Long. del nombre esquema tabla */
   unsigned char   tbschema[SQLUDF_MAX_IDENT_LEN]; /* Nombre del esquema de tabla    */
   unsigned short  tbnamelen;                      /* Longitud del nombre de la tabla*/
   unsigned char   tbname[SQLUDF_MAX_IDENT_LEN];   /* Nombre de la tabla             */
   unsigned short  colnamelen;                     /* Longitud del nombre de columna */
   unsigned char   colname[SQLUDF_MAX_IDENT_LEN];  /* Nombre de columna              */
   unsigned char   ver_rel[SQLUDF_SH_IDENT_LEN];   /* Versión/release de base datos  */
   unsigned char   resd0[2];                       /* Alineación                     */
   sqluint32       platform;                       /* Plataforma                     */
   unsigned short  numtfcol;                       /* # de entradas de la matriz de  */
                                                   /* la lista de columnas TF        */
   unsigned char   resd1[2];                       /* Reservado                      */
   sqluint32       procid;                         /* ID de procedimiento actual     */
   unsigned char   resd2[32];                      /* Reservado                      */
   unsigned short  *tfcolumn;                      /* Se debe asignar la Tfcolumn    */
                                                   /* dinámicamente si se define     */
                                                   /* una función de tabla;          */
                                                   /* si no, un puntero a Null       */
   char            *appl_id;                       /* Identificador de la aplicación */
   sqluint32       dbpartitionnum;                 /* Número de partición de la base */
                                                   /* datos donde se ejecutó rutina  */
   unsigned char   resd3[16];                      /* Reservado                      */
};
```

Aunque no todos los campos de la estructura dbinfo puedan ser útiles durante de una rutina, varios de los valores en los campos de esta estructura serán de utilidad al formular la información sobre los mensajes de errores de diagnóstico. Por ejemplo, si ocurre un error dentro de una rutina, puede que resulte útil devolver el nombre de la base de datos, la longitud del nombre de la base de datos, la página de códigos de la base de datos, el ID de la autorización actual y la longitud del ID de la autorización actual.

Para hacer referencia a la estructura sqludf_dbinfo en una implementación de la rutina LANGUAGE C:

- Añada la cláusula DBINFO a la sentencia CREATE que define la rutina.
- Incluya el archivo de cabecera sqludf.h al principio del archivo que contenga la implementación de la rutina.
- Añada un parámetro de tipo sqludf_dbinfo a la signatura de la rutina en la posición especificada por el estilo de parámetro utilizado.

A continuación figura un ejemplo de un procedimiento C con PARAMETER STYLE GENERAL que muestra el uso de la estructura dbinfo. A continuación se proporciona la sentencia CREATE PROCEDURE para el procedimiento. Observe que según se especifique en la cláusula EXTERNAL NAME, la implementación del procedimiento se encuentra en un archivo de bibliotecas denominado spserver que contiene una función C denominada DbinfoExample:

```
CREATE PROCEDURE DBINFO_EXAMPLE (IN job CHAR(8),
                                 OUT salary DOUBLE,
                                 OUT dbname CHAR(128),
                                 OUT dbversion CHAR(8),
                                 OUT errorcode INTEGER)
DYNAMIC RESULT SETS 0
LANGUAGE C
PARAMETER STYLE GENERAL
DBINFO
FENCED NOT THREADSAFE
READS SQL DATA
PROGRAM TYPE SUB
EXTERNAL NAME 'spserver!DbinfoExample'@
```

A continuación figura la implementación del procedimiento C correspondiente a la definición del procedimiento:

```
/*****************************************************************
 Rutina:    DbinfoExample

   IN:       inJob         - un tipo de trabajo, utilizado en un predicado
                             SELECT
   OUT:      salary        - salario medio de los empleados con trabajo
                             job injob
             dbname        - nombre de base de datos recuperada de DBINFO
             dbversion     - versión de base de datos recuperada de DBINFO
             outSqlError   - sqlcode de error emitido (si los hay)
             sqludf_dbinfo - puntero a la estructura DBINFO

  Finalidad: esta rutina toma un tipo de trabajo y devuelve el
             salario medio de todos los empleados con dicho trabajo,
             así como información sobre la base de datos (nombre,
             versión de la base de datos). La información de la base
             de datos se recupera del objeto dbinfo.

  Muestra cómo:
              - definir parámetros IN/OUT en PARAMETER STYLE GENERAL
              - declarar un puntero de parámetro a la estructura dbinfo
              - recuperar valores de la estructura dbinfo
*****************************************************************/
```

```
SQL_API_RC SQL_API_FN DbinfoExample(char inJob[9],
                                    double *salary,
                                    char dbname[129],
                                    char dbversion[9],
                                    sqlint32 *outSqlError,
                                    struct sqludf_dbinfo * dbinfo
                                    )
{
  /* Declare una SQLCA local */
  struct sqlca sqlca;

  EXEC SQL WHENEVER SQLERROR GOTO return_error;

  /* Sección de declaración de las variables de lenguaje principal de SQL */
  /* Cada nombre de variable del lenguaje principal debe ser exclusivo dentro
     de un archivo de código, si no el precompilador emitirá el error SQL0307 */
  EXEC SQL BEGIN DECLARE SECTION;
  char dbinfo_injob[9];
  double dbinfo_outsalary;
  sqlint16 dbinfo_outsalaryind;
  EXEC SQL END DECLARE SECTION;

  /* Inicializar parámetros de salida - establecer series en NULL */
  memset(dbname, '\0', 129);
  memset(dbversion, '\0', 9);
  *outSqlError = 0;

  /* Copie parámetro de entrada en la variable del lenguaje local */
  strcpy(dbinfo_injob, inJob);

  EXEC SQL SELECT AVG(salary) INTO:dbinfo_outsalary
             FROM employee
               WHERE job =:dbinfo_injob;

  *salary = dbinfo_outsalary;

  /* Copie los valores de la estructura DBINFO en los parámetros de salida.
     Las series deben terminar de forma explícita en nulo.
     La información, como por ejemplo, el nombre de la base de datos y la
     versión del producto de la base de datos, puede encontrarse en la
     estructura DBINFO, así como otros campos de información. */

  strncpy(dbname, (char *)(dbinfo->dbname), dbinfo->dbnamelen);
  dbname[dbinfo->dbnamelen] = '\0';
  strncpy(dbversion, (char *)(dbinfo->ver_rel), 8);
  dbversion[8] = '\0';

  return 0;

  /* Copie el SQLCODE en el parámetro OUT si se produce un error SQL */

  return_error:
  {
    *outSqlError = SQLCODE;
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    return 0;
  }
} /* función DbinfoExample */
```

Área reutilizable como parámetro de función C o C++

La estructura del área reutilizable, utilizada para almacenar valores de UDF entre invocaciones para cada valor de entrada de UDF, recibe soporte en rutinas C y C++ mediante el uso de la estructura sqludf_scrat. Esta estructura C está definida en el archivo de inclusión de DB2 sqludf.h.

Para hacer referencia a la estructura sqludf_scrat, incluya el archivo de cabecera sqludf.h al principio del archivo que contiene la implementación de la función C o C++ y utilice la macro SQLUDF_TRAIL_ARGS_ALL dentro de la signatura de la implementación de la rutina.

En el siguiente ejemplo se hace una demostración de la implementación de una función escalar C que incluye un parámetro de tipo SQLUDF_TRAIL_ARGS_ALL:

```
#ifdef __cplusplus
extern "C"
#endif
void SQL_API_FN ScratchpadScUDF(SQLUDF_INTEGER *outCounter,
                                SQLUDF_SMALLINT *counterNullInd,
                                SQLUDF_TRAIL_ARGS_ALL)
{
  struct scalar_scratchpad_data *pScratData;

  /* SQLUDF_CALLT y SQLUDF_SCRAT son */
  /* partes de SQLUDF_TRAIL_ARGS_ALL */

  pScratData = (struct scalar_scratchpad_data *)SQLUDF_SCRAT->data;
  switch (SQLUDF_CALLT)
  {
    case SQLUDF_FIRST_CALL:
      pScratData->counter = 1;
      break;
    case SQLUDF_NORMAL_CALL:
      pScratData->counter = pScratData->counter + 1;
      break;
    case SQLUDF_FINAL_CALL:
      break;
  }

  *outCounter = pScratData->counter;
  *counterNullInd = 0;
} /* ScratchpadScUDF */
```

La macro SQLUDF_TRAIL_ARGS_ALL se expande para definir otros valores de parámetros, incluido uno que se llama SQLUDF_SCRAT y define un parámetro de memoria intermedia que se debe usar a modo de área reutilizable. Cuando se invoca la función escalar para un conjunto de valores, cada vez que se invoca la función escalar, la memoria intermedia se pasa como parámetro a la función. La memoria intermedia puede servir para acceder a ella

El valor de la macro SQLUDF_TRAIL_ARGS_ALL también define otro parámetro, SQLUDF_CALLT. Este parámetro sirve para indicar un valor de tipo de llamada. Los valores de tipo de llamada pueden servir para identificar si una función se invoca por primera vez para un conjunto de valores, por última vez o en un momento intermedio del proceso.

Soporte de MAIN de tipo de programa para procedimientos C y C++

Aunque generalmente se recomienda el valor por omisión SUB de la cláusula PROGRAM TYPE para procedimientos C, el valor MAIN de la cláusula PROGRAM TYPE recibe soporte en sentencias CREATE PROCEDURE, donde el valor de la cláusula LANGUAGE es C.

El valor MAIN de la cláusula PROGRAM TYPE es necesario para rutinas con más de noventa parámetros.

Cuando se especifica la cláusula PROGRAM TYPE MAIN, los procedimientos deben implementarse utilizando una signatura que sea coherente con el estilo por omisión correspondiente a una rutina principal en una archivo de código de fuente C. Esto no significa que la rutina tenga que implementarse utilizando una función denominada `main`, sino que los parámetros tienen que pasarse en el formato que se asocia generalmente a una implementación de aplicación de la rutina main con tipo por omisión que utilice los argumentos de programación `argc` y `argv` típicos de C.

A continuación figura un ejemplo de una signatura de rutinas C o C++ que cumple la especificación PGRAM TYPE MAIN:

```
SQL_API_RC SQL_API_FN functionName(int argc, char **argv)
{
  ...
}
```

El número total de argumentos de la función se especifica mediante el valor de argc. Los valores de los argumentos se pasan como elementos de matriz dentro de la matriz argv. El número y el orden de los argumentos depende del valor de la cláusula PARAMETER STYLE especificada en la sentencia CREATE PROCEDURE.

Por ejemplo, supongamos que tenemos la siguiente sentencia CREATE PROCEDURE para un procedimiento C que deba tener un estilo PROGRAM TYPE MAIN y el PARAMETER STYLE SQL recomendado:

```
CREATE PROCEDURE MAIN_EXAMPLE (IN job CHAR(8),
OUT salary DOUBLE)
SPECIFIC CPP_MAIN_EXAMPLE
DYNAMIC RESULT SETS 0
NOT DETERMINISTIC
LANGUAGE C
PARAMETER STYLE SQL
NO DBINFO
FENCED NOT THREADSAFE
READS SQL DATA
PROGRAM TYPE MAIN
EXTERNAL NAME 'spserver!MainExample'@
```

La implementación de signatura de rutina correspondiente a esta sentencia CREATE PROCEDURE es la siguiente:

```
//*************************************************
//  Procedimiento almacenado: MainExample
//
//  Parámetros SQL:
//     IN:      argv[1] - job    (char[8])
//     OUT:     argv[2] - salary (double)
//*************************************************
SQL_API_RC SQL_API_FN MainExample(int argc, char **argv)
{
  ...
}
```

Puesto que se utiliza PARAMETER STYLE SQL, además de los valores de los parámetros SQL que se pasan en el momento de la invocación del procedimiento, también se pasan a la rutina los parámetros adicionales necesarios para este estilo.

Se puede acceder a los valores de parámetros haciendo referencia al elemento de la matriz argv que interesa dentro del código fuente. Para el ejemplo anterior, los elementos de matriz argc y argv contienen los siguientes valores:

```
argc   : Número de elementos de la matriz argv
argv[0]: Nombre de la función
argv[1]: Valor de parámetro job (char[8], input)
argv[2]: Valor de parámetro salary (double, output)
argv[3]: indicador nulo para parámetro job
argv[4]: indicador nulo para parámetro salary
argv[5]: sqlstate (char[6], output)
argv[6]: qualName (char[28], output)
argv[7]: specName (char[19], output)
argv[8]: diagMsg (char[71], output)
```

Tipos de datos de SQL soportados en rutinas C y C++

La tabla siguiente lista las correlaciones soportadas entre los tipos de datos de SQL y los tipos de datos de C para las rutinas. Junto a cada tipo de datos de C/C++, se encuentra el tipo definido correspondiente de sqludf.h.

Tabla 37. Tipos de datos de SQL correlacionados con declaraciones C/C++

Tipo de columna SQL	Tipo de datos de C/C++	Descripción del tipo de columna de SQL
SMALLINT	sqlint16 SQLUDF_SMALLINT	Entero con signo de 16 bits
INTEGER	sqlint32 SQLUDF_INTEGER	Entero con signo de 32 bits
BIGINT	sqlint64 SQLUDF_BIGINT	Entero con signo de 64 bits
REAL FLOAT(*n*) donde 1<=n<=24	float SQLUDF_REAL	Coma flotante de precisión simple
DOUBLE FLOAT FLOAT(*n*) donde 25<=n<=53	double SQLUDF_DOUBLE	Coma flotante de precisión doble
DECIMAL(*p*, *s*)	No soportado.	Para pasar un valor decimal, defina el parámetro como tipo de datos convertible desde DECIMAL (por ejemplo, CHAR o DOUBLE) y convierta explícitamente el argumento a este tipo.
CHAR(*n*)	char[*n+1*] donde n es suficientemente grande para contener los datos 1<=*n*<=254 SQLUDF_CHAR	Serie de caracteres de longitud fija terminada en nulo
CHAR(*n*) FOR BIT DATA	char[*n*] donde n es suficientemente grande para contener los datos 1<=*n*<=254 SQLUDF_CHAR	Serie de caracteres de longitud fija no terminada en nulo

Tabla 37. Tipos de datos de SQL correlacionados con declaraciones C/C++ (continuación)

Tipo de columna SQL	Tipo de datos de C/C++	Descripción del tipo de columna de SQL
VARCHAR(*n*)	char[*n+1*] donde n es suficientemente grande para contener los datos 1<=*n*<=32672 SQLUDF_VARCHAR	Serie de longitud variable terminada en nulo

VARCHAR(*n*) FOR BIT DATA	struct {
	sqluint16 length;
	char[*n*]
	}
	
	1<=*n*<=32672
	
	SQLUDF_VARCHAR_FBD
		Serie de caracteres de longitud variable no terminada en nulo

LONG VARCHAR	struct {
	sqluint16 length;
	char[*n*]
	}
	
	1<=*n*<=32700
	
	SQLUDF_LONG
		Serie de caracteres de longitud variable no terminada en nulo

CLOB(*n*)	struct {
	sqluint32 length;
	char data[*n*];
	}
	
	1<=*n*<=2147483647
	
	SQLUDF_CLOB
		Serie de caracteres de longitud variable no terminada en nulo con indicador de longitud de serie de 4 bytes

BLOB(*n*)	struct {
	sqluint32 length;
	char data[n];
	}
	
	1<=*n*<=2147483647
	
	SQLUDF_BLOB
		Serie binaria de longitud variable no terminada en nulo con indicador de longitud de serie de 4 bytes

DATE	char[11] SQLUDF_DATE	Serie de caracteres terminada en nulo con el formato siguiente: `aaaa-mm-dd`
TIME	char[9] SQLUDF_TIME	Serie de caracteres terminada en nulo con el formato siguiente: `hh.mm.ss`
TIMESTAMP	char[27] SQLUDF_STAMP	Serie de caracteres terminada en nulo con el formato siguiente: `aaaa-mm-dd-hh.mm.ss.nnnnnn`

Tabla 37. Tipos de datos de SQL correlacionados con declaraciones C/C++ (continuación)

Tipo de columna SQL	Tipo de datos de C/C++	Descripción del tipo de columna de SQL
LOB LOCATOR	sqluint32 SQLUDF_LOCATOR	Entero con signo de 32 bits
GRAPHIC(*n*)	sqldbchar[*n+1*] donde n es suficientemente grande para contener los datos 1<=*n*<=127 SQLUDF_GRAPH	Serie de caracteres de doble byte y longitud fija terminada en nulo
VARGRAPHIC(*n*)	sqldbchar[*n+1*] donde n es suficientemente grande para contener los datos 1<=*n*<=16336 SQLUDF_GRAPH	Serie de caracteres de doble byte y longitud variable terminada en nulo

LONG VARGRAPHIC	struct {
	sqluint16 length;
	sqldbchar[*n*]
	}
	
	1<=*n*<=16350
	
	SQLUDF_LONGVARG
		Serie de caracteres de doble byte y longitud variable no terminada en nulo

DBCLOB(*n*)	struct {
	sqluint32 length;
	sqldbchar data[n];
	}
	
	1<=*n*<=1073741823
	
	SQLUDF_DBCLOB
		Serie de caracteres de longitud variable no terminada en nulo con indicador de longitud de serie de 4 bytes

XML AS CLOB	struct {
	sqluint32 length;
	char data[*n*];
	}
	
	1<=*n*<=2147483647
	
	SQLUDF_CLOB
		Serie de caracteres serializados de longitud variable no terminada en nulo con indicador de longitud de serie de 4 bytes.

Nota: Los tipos de datos XML solo se pueden implementar como tipos de datos CLOB en rutinas externas implementadas en C o C++.

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

- GRAPHIC(*n*)
- VARGRAPHIC(*n*)
- LONG VARGRAPHIC

- DBCLOB(*n*)

Nota: Se sigue ofreciendo soporte para los tipos de datos LONG VARCHAR y LONG VARGRAPHIC, pero éstos se han dejado de utilizar y no se recomiendan ya que puede que se eliminen en un release futuro.

Manejo de tipos de datos de SQL en rutinas C y C++

En este apartado se identifican los tipos válidos para parámetros y resultados de rutinas y se especifica cómo se debe definir el argumento correspondiente en la rutina, en los lenguajes C o C++. Todos los argumentos de la rutina se deben pasar como punteros al tipo de datos adecuado. Observe que, si utiliza el archivo de inclusión `sqludf.h` y los tipos definidos en éste, puede generar automáticamente estructuras y variables del lenguaje que sean correctas para los distintos tipos de datos y compiladores. Por ejemplo, para BIGINT puede utilizar el tipo de datos SQLUDF_BIGINT a fin de ocultar las diferencias en el tipo requerido para la representación de BIGINT entre distintos compiladores.

El que gobierna el formato de los valores de argumento es el tipo de datos para cada parámetro definido en la sentencia CREATE de la rutina. A fin de obtener el valor en el formato adecuado, puede ser necesario realizar promociones a partir del tipo de datos del argumento. DB2 lleva a cabo automáticamente estas promociones sobre los valores de los argumentos. No obstante, si se especifican tipos de datos incorrectos en el código de la rutina, se producirá un comportamiento imprevisible, como, por ejemplo, pérdida de datos o terminaciones anómalas.

Para el resultado de un método o de una función escalar, es el tipo de datos especificado en la cláusula CAST FROM de la sentencia CREATE FUNCTION el que define el formato. Si no hay ninguna cláusula CAST FROM presente, define el formato el tipo de datos especificado en la cláusula RETURNS.

En el ejemplo siguiente, la presencia de la cláusula CAST FROM significa que el cuerpo de la rutina devuelve un SMALLINT y que DB2 convierte el valor a INTEGER antes de pasarlo a la sentencia en que se produce la referencia a la función:

```
... RETURNS INTEGER CAST FROM SMALLINT ...
```

En este caso, la rutina se debe escribir de forma que genere un SMALLINT, tal como se define más adelante en este apartado. Tenga en cuenta que el tipo de datos CAST FROM se debe *poder convertir* al tipo de datos de RETURNS, por lo que no es posible elegir arbitrariamente otro tipo de datos.

A continuación se muestra una lista de los tipos de SQL y sus representaciones en los lenguajes C/C++. Esta lista incluye información sobre si cada tipo es válido como parámetro o como resultado. También se incluyen ejemplos de cómo pueden aparecer los tipos como definición de argumentos en la rutina, en los lenguajes C o C++:

- SMALLINT

  Válidos. Se representan en C como `SQLUDF_SMALLINT` o `sqlint16`.

  Ejemplo:

  ```
  sqlint16    *arg1;          /* ejemplo de SMALLINT */
  ```

Cuando defina parámetros enteros de una rutina, considere la posibilidad de utilizar INTEGER en lugar de SMALLINT, ya que DB2 no promociona los argumentos INTEGER a SMALLINT. Por ejemplo, suponga que define una UDF del modo siguiente:

```
CREATE FUNCTION SIMPLE(SMALLINT)...
```

Si se invoca la función SIMPLE utilizando datos INTEGER, (`... SIMPLE(1)...`), recibirá un error SQLCODE -440 (SQLSTATE 42884) indicando que no se ha encontrado la función, y es posible que los usuarios finales de esta función no perciban la razón del mensaje. En el ejemplo anterior, 1 es un INTEGER, por lo que puede convertirlo a SMALLINT o puede definir el parámetro como INTEGER.

- INTEGER o INT

  Válidos. Se representan en C como `SQLUDF_INTEGER` o `sqlint32`. Debe ejecutar `#include sqludf.h` o `#include sqlsystm.h` para tomar esta definición.

  Ejemplo:

  ```
  sqlint32 *arg2;          /* ejemplo de INTEGER */
  ```

- BIGINT

  Válidos. Se representan en C como `SQLUDF_BIGINT` o `sqlint64`.

  Ejemplo:

  ```
  sqlint64 *arg3;          /* ejemplo de INTEGER */
  ```

  DB2 define el tipo `sqlint64` del lenguaje C para superar las diferencias entre definiciones del entero con signo de 64 bits en compiladores y sistemas operativos. Debe ejecutar `#include sqludf.h` o `#include sqlsystm.h` para tomar la definición.

- REAL o FLOAT(*n*) donde 1 <= *n* <= 24

  Válidos. Se representan en C como `SQLUDF_REAL` o `float`.

  Ejemplo:

  ```
  float *result;          /* ejemplo de REAL */
  ```

- DOUBLE o DOUBLE PRECISION o FLOAT o FLOAT(*n*) donde 25 <= *n* <= 53

  Válidos. Se representan en C como `SQLUDF_DOUBLE` o `double`.

  Ejemplo:

  ```
  double  *result;        /* ejemplo de DOUBLE   */
  ```

- DECIMAL(p,s) o NUMERIC(p,s)

  No válidos, puesto que no existe ninguna representación en lenguaje C. Si desea pasar un valor decimal, debe definir el parámetro como de tipo de datos DECIMAL que se puede convertir (por ejemplo, CHAR o DOUBLE) y convertir explícitamente el argumento a este tipo. En el caso de DOUBLE, no es necesario que convierta explícitamente un argumento decimal a un parámetro DOUBLE, ya que DB2 lo promociona automáticamente.

  Ejemplo:

  Suponga que tiene dos columnas, WAGE como DECIMAL(5,2) y HOURS como DECIMAL(4,1), y que desea escribir una UDF para calcular el pago semanal en base al salario, el número de horas trabajadas y algunos otros factores. La UDF podría ser como la siguiente:

  ```
  CREATE FUNCTION WEEKLY_PAY (DOUBLE, DOUBLE, ...)
         RETURNS DECIMAL(7,2) CAST FROM DOUBLE
         ...;
  ```

  Para la UDF anterior, los dos primeros parámetros corresponden al salario y al número de horas. Invoque a la UDF WEEKLY_PAY en la sentencia de selección de SQL, del modo siguiente:

```
SELECT WEEKLY_PAY (WAGE, HOURS, ...) ...;
```

Observe que no se requiere ninguna conversión explícita porque los argumentos DECIMAL se pueden convertir a DOUBLE.

Alternativamente, puede definir WEEKLY_PAY con argumentos CHAR, del modo siguiente:

```
CREATE FUNCTION WEEKLY_PAY (VARCHAR(6), VARCHAR(5), ...)
       RETURNS DECIMAL (7,2) CAST FROM VARCHAR(10)
       ...;
```

La puede invocar así:

```
SELECT WEEKLY_PAY (CHAR(WAGE), CHAR(HOURS), ...) ...;
```

Observe que se requiere una conversión explícita porque los argumentos DECIMAL no se pueden promocionar a VARCHAR.

Una ventaja de utilizar parámetros de coma flotante es que resulta fácil realizar operaciones aritméticas sobre los valores de la rutina; una ventaja de utilizar parámetros de tipo carácter es que siempre es posible representar exactamente el valor decimal. Esto no es siempre posible con la coma flotante.

- CHAR(n) o CHARACTER(n) con o sin el modificador FOR BIT DATA.

  Válidos. Se representan en C como `SQLUDF_CHAR` o `char...[n+1]` (ésta es una serie C terminada en nulo).

  Ejemplo:

  ```
  char     arg1[14];      /* ejemplo de CHAR(13)   */
  char    *arg1;          /* también se puede aceptar */
  ```

  Los parámetros de la rutina de entrada del tipo de datos CHAR siempre terminan automáticamente en nulo. Para un parámetro de entrada CHAR(n), donde *n* es la longitud del tipo de datos CHAR, *n* bytes de datos se mueven almacenamiento intermedio en la implementación de la rutina y el carácter situado en la posición *n + 1* se sitúa en el carácter terminador nulo (X'00').

  La rutina debe terminar en nulo de forma explícita los parámetros de salida de los procedimientos y los valores de retorno de las funciones del tipo de datos CHAR. para un valor de retorno de una UDF especificada por la cláusula RETURNS, como por ejemplo RETURNS CHAR(n), o un parámetro de salida de un procedimiento especificado como CHAR(n), donde n es la longitud del valor CHAR, debe existir un terminador nulo dentro de los primeros *n+1* bytes del almacenamiento intermedio. Si se encuentra un terminador nulo dentro de los primeros *n+1* bytes del almacenamiento intermedio, los restantes bytes, hasta el byte *n*, se establecen en los caracteres en blanco de ASCII X'20'). Si no se encuentra ningún terminador nulo, se produce un error de SQL (SQLSTATE 39501).

  Para los parámetros de entrada y salida de los valores de retorno de procedimientos o funciones del tipo de datos CHAR que también especifican la cláusula FOR BIT DATA, que indica que los datos se deben manipular en su formato binario, no se utilizan terminadores nulos para indicar el final del valor del parámetro. Para el valor de retorno de la función RETURNS CHAR(*n*) FOR BIT DATA o para un parámetro de salida de CHAR(*n*) FOR BIT DATA, los *n* primeros bytes del almacenamiento intermedio se copian independientemente de las ocurrencias de los terminados nulos de serie dentro de los *n* primeros bytes. Los caracteres de terminador nulo identificados dentro del almacenamiento intermedio se pasan por alto como terminados nulos y simplemente se tratan como datos normales.

  Tenga precaución al utilizar las funciones normales de manejo de series C en una rutina que manipule un valor FOR BIT DATA, ya que muchas de estas funciones buscan un terminador nulo para delimitar un argumento de serie y los terminadores nulos (X'00') pueden aparecer legítimamente en medio de un valor

FOR BIT DATA. El uso de las funciones C en los valores FOR BIT DATA puede causar el truncamiento no deseado del valor de datos.

Cuando defina parámetros de tipo carácter de una rutina, piense en la posibilidad de utilizar VARCHAR en lugar de CHAR, puesto que DB2 no promociona los argumentos VARCHAR a CHAR y los literales de serie se consideran automáticamente VARCHAR. Por ejemplo, suponga que define una UDF del modo siguiente:

```
CREATE FUNCTION SIMPLE(INT,CHAR(1))...
```

Si se invoca la función SIMPLE utilizando datos VARCHAR, (`... SIMPLE(1,'A')...`), recibirá un error SQLCODE -440 (SQLSTATE 42884) indicando que no se ha encontrado la función, y es posible que los usuarios finales de esta función no perciban la razón del mensaje. En el ejemplo anterior, `'A'` es VARCHAR, por lo que puede convertirlo a CHAR o puede definir el parámetro como VARCHAR.

- VARCHAR(n) FOR BIT DATA o LONG VARCHAR con o sin el modificador FOR BIT DATA.

  Válidos. Representan VARCHAR(n) FOR BIT DATA en C como `SQLUDF_VARCHAR_FBD`. Representan LONG VARCHAR en C como `SQLUDF_LONG`. De lo contrario, representan estos dos tipos de SQL en C como una estructura similar a la siguiente del archivo de inclusión sqludf.h:

  ```
  struct sqludf_vc_fbd
  {
     unsigned short length;         /* longitud de los datos */
     char           data[1];        /* primer carácter de datos */
  };
  ```

  El [1] indica una matriz para el compilador. No significa que sólo se pase un carácter; dado que se pasa la dirección de la estructura, y no la estructura real, proporciona una manera de utilizar la lógica de matrices.

  Estos valores no se representan como series C terminadas en nulo porque el carácter de nulo podría ser admisible formando parte del valor de los datos. Se pasa explícitamente a la rutina la longitud de los parámetros utilizando la variable de estructura `length`. Para la cláusula RETURNS, la longitud que se pasa a la rutina es la longitud del almacenamiento intermedio. Lo que el cuerpo de la rutina debe devolver, utilizando la variable de estructura `length`, es la longitud real del valor de los datos.

  Ejemplo:

  ```
  struct sqludf_vc_fbd *arg1;  /* ejemplo de VARCHAR(n) FOR BIT DATA */
  struct sqludf_vc_fbd *result; /* y también de LONG VARCHAR FOR BIT DATA */
  ```

  Nota: Se sigue ofreciendo soporte para los tipos de datos LONG VARCHAR y LONG VARGRAPHIC, pero éstos se han dejado de utilizar y no se recomiendan ya que puede que se eliminen en un release futuro.

- VARCHAR(n) sin FOR BIT DATA.

  Válidos. Se representan en C como `SQLUDF_VARCHAR` o `char...[n+1]`. (Ésta es una serie C terminada en nulo.)

  Para un parámetro VARCHAR(n), DB2 colocará un nulo en la posición (k+1), donde k es la longitud de la serie en particular. Las funciones de manejo de series C son adecuadas para la manipulación de estos valores. Para un valor de RETURNS VARCHAR(n) o un parámetro de salida de un procedimiento almacenado, el cuerpo de la rutina debe delimitar el valor real con un nulo porque DB2 determinará la longitud del resultado a partir de este carácter de nulo.

  Ejemplo:

```
char     arg2[51];      /* ejemplo de VARCHAR(50)   */
char    *result;        /* también se puede aceptar */
```

- DATE

  Válidos. Se representan en C como SQLUDF_DATE o CHAR(10), es decir, como char...[11]. El valor de fecha siempre se pasa a la rutina con formato ISO:

  ```
  aaaa-mm-dd
  ```

  Ejemplo:

  ```
  char     arg1[11];      /* ejemplo de DATE       */
  char    *result;        /* también se puede aceptar */
  ```

  Nota: Para los valores de retorno de DATE, TIME y TIMESTAMP, DB2 exige que los caracteres estén en el formato definido y, si no es así, DB2 puede malinterpretar el valor (por ejemplo, 2001-04-03 se interpretaría como el 3 de abril, aunque la intención es que interprete el 4 de marzo) o generaría un error (SQLCODE -493, SQLSTATE 22007).

- TIME

  Válidos. Se representan en C como SQLUDF_TIME o CHAR(8), es decir, como char...[9]. El valor de hora siempre se pasa a la rutina con formato ISO:

  ```
  hh.mm.ss
  ```

  Ejemplo:

  ```
  char    *arg;           /* ejemplo para TIME          */
  char     result[9];     /* también se puede aceptar */
  ```

- TIMESTAMP

  Válidos. Se representan en C como SQLUDF_STAMP o CHAR(26), es decir, como char...[27]. El valor de indicación de la hora siempre se pasa con el formato siguiente:

  ```
  aaaa-mm-dd-hh.mm.ss.nnnnnn
  ```

  Ejemplo:

  ```
  char     arg1[27];      /* ejemplo de TIMESTAMP  */
  char    *result;        /* también se puede aceptar */
  ```

- GRAPHIC(n)

  Válidos. Se representan en C como SQLUDF_GRAPH o sqldbchar[n+1]. (Ésta es una serie gráfica terminada en nulo.) Tenga en cuenta que puede utilizar wchar_t[n+1] en los sistemas operativos en los que wchar_t está definido con 2 bytes de longitud; no obstante, es recomendable sqldbchar.

  Para un parámetro GRAPHIC(n), DB2 mueve *n* caracteres de doble byte al almacenamiento intermedio y establece los dos bytes siguientes en nulo. Los datos que se pasan de DB2 a una rutina están en formato DBCS y es de esperar que el resultado devuelto esté en formato DBCS. Este comportamiento es el mismo que si se utiliza la opción de precompilador WCHARTYPE NOCONVERT. Para un valor de RETURNS GRAPHIC(*n*) o un parámetro de salida de un procedimiento almacenado, DB2 busca un CHAR GRAPHIC nulo incorporado y, si lo encuentra, rellena el valor hasta *n* con caracteres GRAPHIC en blanco.

  Cuando defina parámetros gráficos de una rutina, considere la posibilidad de utilizar VARGRAPHIC en lugar de GRAPHIC, ya que DB2 no promociona los argumentos VARGRAPHIC a GRAPHIC. Por ejemplo, suponga que define una rutina del modo siguiente:

  ```
  CREATE FUNCTION SIMPLE(GRAPHIC)...
  ```

  Si se invoca la función SIMPLE utilizando datos VARGRAPHIC data, (... SIMPLE('*literal_gráfico*')...), recibirá un error SQLCODE -440 (SQLSTATE 42884) indicando que no se ha encontrado la función, y es posible

que los usuarios finales de esta función no comprendan la razón de este mensaje. En el ejemplo anterior, *literal_gráfico* es una serie DBCS literal que se interpreta como datos VARGRAPHIC, por lo que puede convertirla a GRAPHIC o puede definir el parámetro como VARGRAPHIC.

Ejemplo:

```
sqldbchar   arg1[14];      /* ejemplo de GRAPHIC(13)   */
sqldbchar  *arg1;          /* también se puede aceptar */
```

- VARGRAPHIC(n)

  Válidos. Se representan en C como `SQLUDF_GRAPH` o `sqldbchar[n+1]`. (Ésta es una serie gráfica terminada en nulo.) Tenga en cuenta que puede utilizar `wchar_t[n+1]` en los sistemas operativos en los que `wchar_t` está definido con 2 bytes de longitud; no obstante, es recomendable `sqldbchar`.

  Para un parámetro VARGRAPHIC(*n*), DB2 colocará un nulo gráfico en la posición (k+1), donde *k* es la longitud de la aparición en particular. Un nulo gráfico hace referencia a una situación en que todos los bytes del último carácter de la serie gráfica contienen ceros binarios ('\0's). Los datos que se pasan de DB2 a una rutina están en formato DBCS y es de esperar que el resultado devuelto esté en formato DBCS. Este comportamiento es el mismo que si se utiliza la opción de precompilador WCHARTYPE NOCONVERT. Para un valor de RETURNS VARGRAPHIC(*n*) o un parámetro de salida de un procedimiento almacenado, el cuerpo de la rutina debe delimitar el valor real con un nulo gráfico, porque DB2 determinará la longitud del resultado a partir de este carácter de nulo gráfico.

  Ejemplo:

```
sqldbchar   args[51],    /* ejemplo de VARGRAPHIC(50) */
sqldbchar   *result,     /* también se puede aceptar  */
```

- LONG VARGRAPHIC

  Válidos. Se representan en C como `SQLUDF_LONGVARG` o como una estructura:

```
struct sqludf_vg
{
   unsigned short length;         /* longitud de los datos */
   sqldbchar      data[1];        /* primer carácter de datos */
};
```

  Tenga en cuenta que, en la estructura anterior, puede utilizar `wchar_t` en lugar de `sqldbchar` en los sistemas operativos en los que `wchar_t` está definido con 2 bytes de longitud; no obstante, es recomendable el uso de `sqldbchar`.

  El [1] simplemente indica una matriz para el compilador. No significa que sólo se pase un carácter gráfico. Dado que se pasa la dirección de la estructura, y no la estructura real, se proporciona una manera de utilizar la lógica de matrices.

  No se representan como series gráficas terminadas en nulo. Se pasa explícitamente a la rutina la longitud de los parámetros, en caracteres de doble byte, utilizando la variable de estructura `length`. Los datos que se pasan de DB2 a una rutina están en formato DBCS y es de esperar que el resultado devuelto esté en formato DBCS. Este comportamiento es el mismo que si se utiliza la opción de precompilador WCHARTYPE NOCONVERT. Para la cláusula RETURNS o un parámetro de salida de un procedimiento almacenado, la longitud que se pasa a la rutina es la longitud del almacenamiento intermedio. Lo que el cuerpo de la rutina debe devolver, utilizando la variable de estructura `length`, es la longitud real del valor de los datos, en caracteres de doble byte.

  Ejemplo:

```
struct sqludf_vg *arg1;  /* ejemplo de VARGRAPHIC(n)   */
struct sqludf_vg *result; /* y también de LONG VARGRAPHIC   */
```

Nota: Se sigue ofreciendo soporte para los tipos de datos LONG VARCHAR y LONG VARGRAPHIC, pero éstos se han dejado de utilizar y no se recomiendan ya que puede que se eliminen en un release futuro.

- BLOB(n) y CLOB(n)

  Válidos. Se representan en C como SQLUDF_BLOB, SQLUDF_CLOB, o como una estructura:

  ```
  struct sqludf_lob
  {
      sqluint32     length;      /* longitud en bytes */
      char          data[1];      /* primer byte del lob */
  };
  ```

  El [1] simplemente indica una matriz para el compilador. No significa que sólo se pase un carácter; dado que se pasa la dirección de la estructura, y no la estructura real, proporciona una manera de utilizar la lógica de matrices.

  No se representan como series C terminadas en nulo. Se pasa explícitamente a la rutina la longitud de los parámetros utilizando la variable de estructura `length`. Para la cláusula RETURNS o un parámetro de salida de un procedimiento almacenado, la longitud que se devuelve a la rutina es la longitud del almacenamiento intermedio. Lo que el cuerpo de la rutina debe devolver, utilizando la variable de estructura `length`, es la longitud real del valor de los datos.

  Ejemplo:

  ```
  struct sqludf_lob *arg1;  /* ejemplo de BLOB(n), CLOB(n) */
  struct sqludf_lob *result;
  ```

- DBCLOB(n)

  Válidos. Se representan en C como SQLUDF_DBCLOB o como una estructura:

  ```
  struct sqludf_lob
  {
      sqluint32 length;      /* longitud en caracteres gráficos */
      sqldbchar data[1];         /* primer byte del lob */
  };
  ```

  Tenga en cuenta que, en la estructura anterior, puede utilizar `wchar_t` en lugar de `sqldbchar` en los sistemas operativos en los que `wchar_t` está definido con 2 bytes de longitud; no obstante, es recomendable el uso de `sqldbchar`.

  El [1] simplemente indica una matriz para el compilador. No significa que sólo se pase un carácter gráfico; dado que se pasa la dirección de la estructura, y no la estructura real, proporciona una manera de utilizar la lógica de matrices.

  No se representan como series gráficas terminadas en nulo. Se pasa explícitamente a la rutina la longitud de los parámetros utilizando la variable de estructura `length`. Los datos que se pasan de DB2 a una rutina están en formato DBCS y es de esperar que el resultado devuelto esté en formato DBCS. Este comportamiento es el mismo que si se utiliza la opción de precompilador WCHARTYPE NOCONVERT. Para la cláusula RETURNS o un parámetro de salida de un procedimiento almacenado, la longitud que se pasa a la rutina es la longitud del almacenamiento intermedio. Lo que el cuerpo de la rutina debe devolver, utilizando la variable de estructura `length`, es la longitud real del valor de los datos, con todas estas longitudes expresadas en caracteres de doble byte.

  Ejemplo:

  ```
  struct sqludf_lob *arg1;  /* ejemplo de DBCLOB(n) */
  struct sqludf_lob *result;
  ```

- Tipos diferenciados

  Válidos o no válidos en función del tipo base. Los tipos diferenciados se pasarán a la UDF con el formato del tipo base del UDT, por lo que se pueden especificar si y sólo si es válido el tipo base.

Ejemplo:

```
struct sqludf_lob *arg1; /* para tipos diferenciados basados en BLOB(n) */
double            *arg2; /* para tipos diferenciados basados en DOUBLE  */
char              res[5];/* para tipos diferenciados basados en CHAR(4) */
```

- XML

  Válidos. Se representa en C como SQLUDF_XML o como se representa el tipo de datos CLOB; es decir, con una estructura:

  ```
  struct sqludf_lob
  {
      sqluint32     length;      /* longitud en bytes */
      char          data[1];      /* primer byte del lob */
  };
  ```

  El [1] simplemente indica una matriz para el compilador. No significa que sólo se pase un carácter; dado que se pasa la dirección de la estructura, y no la estructura real, proporciona una manera de utilizar la lógica de matrices.

  No se representan como series C terminadas en nulo. Se pasa explícitamente a la rutina la longitud de los parámetros utilizando la variable de estructura `length`. Para la cláusula RETURNS o un parámetro de salida de un procedimiento almacenado, la longitud que se devuelve a la rutina es la longitud del almacenamiento intermedio. Lo que el cuerpo de la rutina debe devolver, utilizando la variable de estructura `length`, es la longitud real del valor de los datos.

  Ejemplo:

  ```
  struct sqludf_lob *arg1;  /* ejemplo de XML(n) */
  struct sqludf_lob *result;
  ```

  La asignación de valores de parámetros y variables XML y el acceso a ellos en código externo de la rutina C y C++ se hace como para los valores CLOB.

- Tipos diferenciados AS LOCATOR, o cualquier tipo de LOB AS LOCATOR

  Válidos para los parámetros y resultados de UDF y métodos. Sólo se puede utilizar para modificar tipos de LOB o cualquier tipo diferenciado que esté basado en un tipo de LOB. Su representación en C es como SQLUDF_LOCATOR o como entero de cuatro bytes.

  El valor de localizador se puede asignar a cualquier variable del lenguaje principal de localizador que sea de un tipo compatible, y luego se puede utilizar en una sentencia de SQL. Esto significa que las variables de localizador únicamente son de utilidad en las UDF y los métodos definidos con un indicador de acceso a SQL que sea CONTAINS SQL o superior. Para que sean compatibles con las UDF y los métodos existentes, las API de localizador se siguen soportando para las UDF NOT FENCED NO SQL. No se recomienda el uso de estas API para las funciones nuevas.

  Ejemplo:

  ```
  sqludf_locator       *arg1;   /* argumento de localizador  */
  sqludf_locator       *result; /* resultado del localizador */


EXEC SQL BEGIN DECLARE SECTION;
   SQL TYPE IS CLOB LOCATOR arg_loc;
   SQL TYPE IS CLOB LOCATOR res_loc;
EXEC SQL END DECLARE SECTION;

/* Extraer algunos caracteres del medio */
/* del argumento y devolverlos         */
*arg_loc = arg1;
EXEC SQL VALUES SUBSTR(arg_loc, 10, 20) INTO :res_loc;
*result = res_loc;
```

- Tipos estructurados

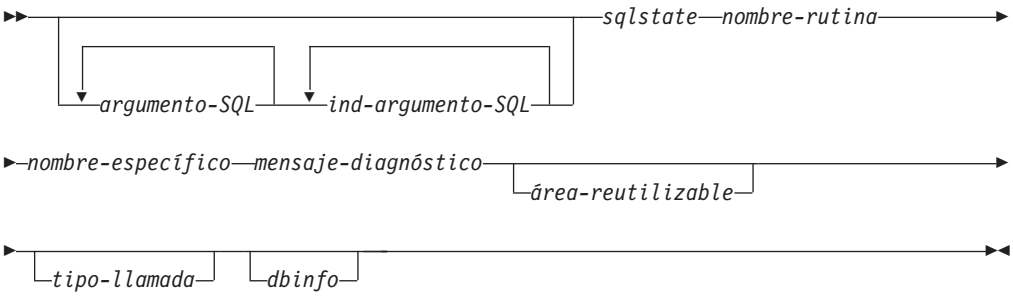
Válidos para los parámetros y resultados de UDF y métodos en que exista una función de transformación apropiada. Los parámetros de tipos estructurados se pasarán a la función o al método en el tipo de resultado de la función de transformación FROM SQL. Los resultados de tipos estructurados se pasarán en el tipo de parámetro de la función de transformación TO SQL.

Paso de argumentos a rutinas C, C++, OLE o COBOL

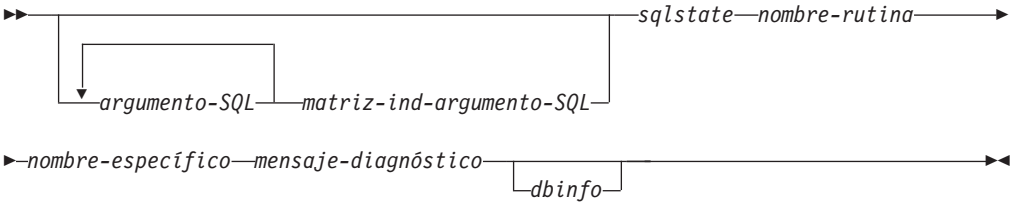
Además de los argumentos de SQL que se especifican en la referencia DML para una rutina, DB2 pasa argumentos adicionales al cuerpo de la rutina externa. La naturaleza y el orden de estos argumentos lo determina el estilo de parámetro con que se ha registrado la rutina. Para asegurarse de que se intercambie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

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

Rutinas PARAMETER STYLE SQL

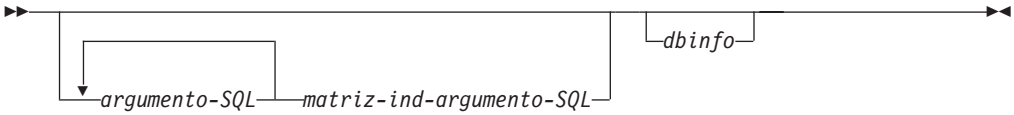


Procedimientos PARAMETER STYLE DB2SQL



Procedimientos PARAMETER STYLE GENERAL

Procedimientos PARAMETER STYLE GENERAL WITH NULLS



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

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

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

- Para una función escalar, un argumento para cada parámetro de entrada para la función, seguidos de un *argumento-SQL* para el resultado de la función.
- Para una función de tabla, un argumento para cada parámetro de entrada para la función, seguidos de un *argumento-SQL* para cada columna de la tabla de resultados de la función.
- Para un método, un *argumento-SQL* para el tipo sujeto del método, luego un argumento para cada parámetro de entrada para el método, seguidos de un *argumento-SQL* para el resultado del método.
- Para un procedimiento almacenado, un *argumento-SQL* para cada parámetro del procedimiento almacenado.

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

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

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

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

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

  Para las funciones de tabla, DB2 define una optimización del rendimiento si no se tienen que devolver a DB2 todas las columnas definidas. Si escribe una UDF que se aproveche de esta característica, ésta sólo devolverá las columnas que necesite la sentencia que hace referencia a la función de tabla. Por ejemplo, piense en una sentencia CREATE FUNCTION para una función de tabla definida con 100 columnas de resultado. Si una sentencia determinada que hace referencia

a la función sólo está interesada en dos de ellas, esta optimización permite que la UDF sólo devuelva estas dos columnas de cada fila, y no invierta tiempo en las otras 98 columnas. Para obtener más información sobre la optimización mencionada, consulte el argumento dbinfo más adelante.

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

- Parámetro INOUT de un procedimiento almacenado

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

DB2 alinea los datos de *argumento-SQL* de acuerdo con el tipo de datos y el sistema operativo, conocido también como plataforma.

ind-argumento-SQL...

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

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

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

  DB2 establece este argumento antes de llamar a la rutina. Contiene uno de los valores siguientes:

  0 El argumento está presente y no es NULL.

  -1 El argumento está presente y su valor es NULL.

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

- Resultado de una función o método, o parámetro OUT de un procedimiento almacenado

  La rutina establece este argumento antes de volver a DB2. La rutina utiliza este argumento para señalar si el valor del resultado en concreto es NULL:

  0 El resultado no es NULL.

  -1 El resultado es el valor NULL.

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

Para los métodos y funciones escalares, DB2 trata un resultado NULL como un error aritmético si se dan las circunstancias siguientes:

– El parámetro de configuración de base de datos *dft_sqlmathwarn* es YES
– Uno de los argumentos de entrada es nulo debido a un error aritmético

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

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

- Parámetro INOUT de un procedimiento almacenado

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

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

matriz-ind-argumento-SQL

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

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

- Parámetro IN de un procedimiento almacenado

  DB2 establece este elemento antes de llamar a la rutina. Contiene uno de los valores siguientes:

  0 El argumento está presente y no es NULL.
  -1 El argumento está presente y su valor es NULL.

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

- Parámetro OUT de un procedimiento almacenado

  La rutina establece este elemento antes de volver a DB2. La rutina utiliza este argumento para señalar si el valor del resultado en concreto es NULL:

  0 o positivo
  El resultado no es NULL.
  negativo
  El resultado es el valor NULL.

- Parámetro INOUT de un procedimiento almacenado

  Este elemento se comporta como los parámetros IN y OUT y, por consiguiente, sigue los dos conjuntos de reglas indicados anteriormente. DB2 establecerá el argumento antes de llamar al procedimiento

almacenado. El procedimiento almacenado establece el elemento de *matriz-ind-argumento-SQL* antes de volver a DB2.

Cada elemento de *matriz-ind-argumento-SQL* toma la forma de un valor SMALLINT. DB2 alinea los datos de *matriz-ind-argumento-SQL* de acuerdo con el tipo de datos y el sistema operativo.

sqlstate La rutina establece este argumento antes de volver a DB2. Ésta lo puede utilizar para señalar condiciones de aviso o error. La rutina puede establecer este argumento en cualquier valor. El valor '00000' significa que no se ha detectado ninguna condición de aviso ni de error. Los valores que empiezan por '01' son condiciones de aviso. Los valores que empiezan por cualquier cosa distinta de '00' y '01' son condiciones de error. Cuando se llama a la rutina, el argumento contiene el valor '00000'.

Para las condiciones de error, la rutina devuelve un SQLCODE de -443. Para las condiciones de aviso, devuelve un SQLCODE de +462. Si el SQLSTATE es 38001 ó 38502, el SQLCODE es -487.

El *sqlstate* toma la forma de un valor CHAR(5). DB2 alinea los datos para *sqlstate* en función del tipo de datos y del sistema operativo del servidor.

nombre-rutina
DB2 establece este argumento antes de llamar a la rutina. Se trata del nombre de función calificado que se pasa de DB2 a la rutina

El formato del *nombre-rutina* que se pasa es el siguiente:

```
esquema.rutina
```

Las distintas partes se separan mediante un punto. Éstos son dos ejemplos:

```
PABLO.BLOOP   WILLIE.FINDSTRING
```

Este formato permite utilizar el mismo cuerpo de rutina para varias rutinas externas, y sigue diferenciando entre las rutinas cuando se las invoca.

Nota: Aunque es posible incluir un punto en nombres de objeto y nombres de esquema, no es aconsejable. Por ejemplo, si una función `ROTATE` se encuentra en un esquema `OBJ.OP`, el nombre de rutina que se pasa a la función es `OBJ.OP.ROTATE`, y no resulta obvio si el nombre de esquema es `OBJ` u `OBJ.OP`.

El *nombre-rutina* adopta el formato de un valor VARCHAR(257). DB2 alinea los datos para *nombre-rutina* en función del tipo de datos y del sistema operativo del servidor.

nombre-específico
DB2 establece este argumento antes de llamar a la rutina. Se trata del nombre específico de la rutina que se pasa de DB2 a la rutina.

Éstos son dos ejemplos:

```
WILLIE_FIND_FEB99 SQL9904281052440430
```

El usuario proporciona este primer valor en su sentencia CREATE. El segundo valor, si el usuario no especifica ninguno, lo genera DB2 a partir de la indicación de la hora actual.

Al igual que con el argumento *nombre-rutina*, la razón para pasar este valor consiste en brindar a la rutina un medio para distinguir exactamente qué rutina concreta lo está invocando.

El *nombre-específico* toma la forma de un valor VARCHAR(18). DB2 alinea los datos para *nombre-específico* en función del tipo de datos y del sistema operativo del servidor.

mensaje-diagnóstico
La rutina establece este argumento antes de volver a DB2. La rutina puede utilizar este argumento para insertar texto de mensaje en un mensaje de DB2.

Cuando la rutina devuelve un error o un aviso, utilizando el argumento *sqlstate* descrito anteriormente, puede incluir aquí información descriptiva. DB2 incluye esta información como señal en su mensaje.

DB2 establece el primer carácter en nulo antes de llamar a la rutina. Al volver, trata la serie como una serie C terminada en nulo. Esta serie se incluirá en la SQLCA como señal de la condición de error. Como mínimo, la primera parte de esta serie aparecerá en la SQLCA o en el mensaje del CLP de DB2. No obstante, el número real de caracteres que aparecerán depende de las longitudes de las otras señales, puesto que DB2 trunca las señales para que se ajusten al límite impuesto por la SQLCA respecto a la longitud total de las señales. Evite utilizar X'FF' en el texto, ya que este carácter se utiliza para delimitar señales en la SQLCA.

La rutina no debe devolver más texto del que quepa en el almacenamiento intermedio de VARCHAR(70) que se le pasa. Una sobregrabación por parte de la rutina puede ocasionar resultados imprevisibles o una terminación anómala.

DB2 supone que las señales de mensajes devueltas a DB2 por la rutina están en la misma página de códigos que la rutina. La rutina se debe asegurar de que es así. Si se utiliza el subconjunto ASCII invariable de 7 bits, la rutina puede devolver las señales de mensajes en cualquier página de códigos.

El *mensaje-diagnóstico* toma la forma de un valor VARCHAR(70). DB2 alinea los datos para *mensaje-diagnóstico* en función del tipo de datos y del sistema operativo del servidor.

área-reutilizable
DB2 establece este argumento antes de invocar la UDF o al método. Sólo está presente para las funciones y los métodos en que se ha especificado la palabra clave SCRATCHPAD durante el registro. Este argumento es una estructura, exactamente igual que la estructura utilizada para pasar un valor de cualquiera de los tipos de datos LOB, con los elementos siguientes:

- Un INTEGER que contiene la longitud del área reutilizable. Si se cambia la longitud del área reutilizable, se producirá un SQLCODE -450 (SQLSTATE 39501)
- El área reutilizable real inicializada completamente con ceros binarios del modo siguiente:
  - Para los métodos y las funciones escalares, se inicializa antes de la primera llamada y DB2 no la suele examinar ni modificar después.
  - Para las funciones de tabla, el área reutilizable se inicializa antes de la primera (FIRST) llamada a la UDF si se especifica FINAL CALL en la sentencia CREATE FUNCTION. A partir de esta llamada, el contenido del área reutilizable está totalmente bajo el control de la función de tabla. Si se ha especificado, o asumido por omisión, NO FINAL CALL para una función de tabla, el área reutilizable se inicializa para cada

llamada OPEN, y el contenido de la misma está totalmente bajo el control de la función de tabla entre llamadas OPEN. (Esto puede ser muy importante para una función de tabla utilizada en una unión o en una subconsulta. Si es necesario mantener el contenido del área reutilizable a lo largo de las llamadas OPEN, se debe especificar FINAL CALL en la sentencia CREATE FUNCTION. Si se especifica FINAL CALL, además de las llamadas OPEN, FETCH y CLOSE normales, la función de tabla recibirá también llamadas FIRST y FINAL, con el objetivo de mantener el área reutilizable y liberar recursos.)

El área reutilizable se puede correlacionar en la rutina utilizando el mismo tipo que un CLOB o un BLOB, puesto que el argumento que se pasa tiene la misma estructura.

Asegúrese de que el código de la rutina no realiza cambios fuera del almacenamiento intermedio del área reutilizable. Una sobregrabación por parte de la rutina puede causar resultados imprevisibles o una terminación anómala y puede que DB2 experimente una anomalía no leve.

Si en una subconsulta se hace referencia a un método o una UDF escalar que emplea un área reutilizable, es posible que DB2 decida renovar el área reutilizable entre invocaciones de la subconsulta. Esta renovación se produce después de realizar una llamada final, en caso de que se haya especificado FINAL CALL para la UDF.

DB2 inicializa el área reutilizable de forma que el campo de datos esté alineado para el almacenamiento de cualquier tipo de datos. Esto puede hacer que la estructura entera del área reutilizable, incluido el campo de longitud, esté alineada incorrectamente.

tipo-llamada
DB2 establece este argumento, si está presente, antes de invocar la UDF o al método. Este argumento está presente para todas las funciones de tabla y para los métodos y funciones escalares en que se ha especificado FINAL CALL durante el registro.

A continuación, se indican todos los valores posibles actuales para *tipo-llamada*. La UDF o el método debe contener una sentencia de conmutación o de caso que pruebe explícitamente todos los valores esperados, en lugar de contener lógica del tipo "if A do AA, else if B do BB, else it must be C so do CC". Esto es así porque puede que en el futuro se añadan tipos de llamadas adicionales y, si no prueba explícitamente la condición C, tendrá problemas cuando se añadan las nuevas posibilidades.

Nota:

1. Para todos los valores de *tipo-llamada*, puede ser conveniente que la rutina establezca un valor de retorno de *sqlstate* y de *mensaje-diagnóstico*. Esta información no se repetirá en las descripciones siguientes de cada uno de los *tipos-llamada*. Para todas las llamadas, DB2 emprenderá la acción indicada descrita anteriormente para estos argumentos.
2. El archivo de inclusión `sqludf.h` está destinado a la utilización con rutinas. El archivo contiene definiciones simbólicas para los valores de *tipo-llamada* siguientes, que se escriben como constantes.

Para los métodos y funciones escalares, *tipo-llamada* contiene:

SQLUDF_FIRST_CALL (-1)
Ésta es la primera (FIRST) llamada a la rutina para esta sentencia.

El *área reutilizable* (si la hay) se establece con ceros binarios cuando se llama a la rutina. Se pasan los valores de todos los argumentos y la rutina debe realizar las acciones puntuales de inicialización que sean necesarias. Además, una llamada FIRST a un método o una UDF escalar es como una llamada NORMAL, en el sentido en que se espera que desarrolle y devuelva una respuesta.

Nota: Si se especifica SCRATCHPAD pero no así FINAL CALL, la rutina no tendrá este argumento de *tipo-llamada* para identificar la llamada realmente primera. En lugar de esto, se tendrá que basar en el estado del área reutilizable, todo ceros.

SQLUDF_NORMAL_CALL (0)
Ésta es una llamada NORMAL. Se pasan todos los valores de entrada de SQL y se espera que la rutina desarrolle y devuelva el resultado. También puede que la rutina devuelva información de *sqlstate* y de *mensaje-diagnóstico*.

SQLUDF_FINAL_CALL (1)
Ésta es una llamada FINAL; es decir, no se pasa ningún valor de *argumento-SQL* ni de *ind-argumento-SQL*, y los intentos de examinar estos valores pueden causar resultados imprevisibles. Si también se pasa un *área reutilizable*, ésta permanece inalterada desde la llamada anterior. Se espera que la rutina libere recursos en este punto.

SQLUDF_FINAL_CRA (255)
Ésta es una llamada FINAL, idéntica a la llamada FINAL descrita antes, con una característica adicional, a saber, que se crea para rutinas que están definidas como capaces de emitir SQL, y se crea en un momento en que la rutina no debe emitir SQL, a excepción de CLOSE cursor. (SQLCODE -396, SQLSTATE 38505) Por ejemplo, cuando DB2 está en medio de un proceso COMMIT, no puede tolerar nuevo SQL, y cualquier llamada FINAL emitida para una rutina en ese momento será una llamada 255 FINAL. Las rutinas que no estén definidas para contener ningún nivel de acceso a SQL nunca recibirán una llamada 255 FINAL, mientras que las rutinas que utilizan SQL pueden recibir cualquier tipo de llamada FINAL.

Liberación de recursos

De un método o una UDF escalar se espera que liberen los recursos que han necesitado, por ejemplo memoria. Si se especifica FINAL CALL para la rutina, esa llamada FINAL es el lugar natural en el que liberar recursos, siempre que también se especifique SCRATCHPAD y se utilice para hacer un seguimiento del recurso. Si no se especifica FINAL CALL, cualquier recurso adquirido se debe liberar en la misma llamada.

Para las funciones de tabla, *tipo-llamada* contiene:

SQLUDF_TF_FIRST (-2)
Ésta es la primera (FIRST) llamada, que sólo se produce si se ha especificado la palabra clave FINAL CALL para la UDF. El *área reutilizable* se establece con ceros binarios antes de esta llamada. Los valores de los argumentos se pasan a la función de tabla. La función de tabla puede adquirir memoria o realizar otra inicialización puntual de sólo recursos. Esto no es una llamada OPEN. Una llamada OPEN sigue a continuación de esta llamada.

En una llamada FIRST, la función de tabla no debe devolver ningún dato a DB2, ya que DB2 hace caso omiso de los datos.

SQLUDF_TF_OPEN (-1)
Ésta es la llamada OPEN. El *área reutilizable* se inicializará si se especifica NO FINAL CALL, pero no necesariamente en caso contrario. En OPEN se pasan los valores de todos los argumentos de SQL a la función de tabla. La función de tabla no debe devolver ningún dato a DB2 en la llamada OPEN.

SQLUDF_TF_FETCH (0)
Ésta es una llamada FETCH y DB2 espera que la función de tabla devuelva una fila que comprenda el conjunto de valores de retorno o una condición de fin de tabla, indicada por el valor '02000' de SQLSTATE. Si se pasa el *área reutilizable* a la UDF, al entrar ésta permanece inalterada respecto a la llamada anterior.

SQLUDF_TF_CLOSE (1)
Ésta es una llamada CLOSE para la función de tabla. Equilibra la llamada OPEN y se puede utilizar para realizar cualquier proceso CLOSE externo (por ejemplo, cerrar un archivo fuente) y para la liberación de recursos (especialmente para el caso de NO FINAL CALL).

En los casos que implican una unión o una subconsulta, se pueden repetir secuencias de llamadas OPEN/FETCH.../CLOSE dentro de la ejecución una sentencia, pero sólo habrá una llamada FIRST y una llamada FINAL. Las llamadas FIRST y FINAL sólo se producen si se especifica FINAL CALL para la función de tabla.

SQLUDF_TF_FINAL (2)
Ésta es una llamada FINAL, que sólo se produce si se ha especificado FINAL CALL para la función de tabla. Equilibra la llamada FIRST y sólo se produce una vez por ejecución de la sentencia. Está destinada a liberar recursos.

SQLUDF_TF_FINAL_CRA (255)
Ésta es una llamada FINAL, idéntica a la llamada FINAL descrita antes, con una característica adicional, digamos que está formada por UDF que están definidas como capaces de emitir SQL, y se ha creado en un momento en que la UDF no debe emitir SQL, a excepción de CLOSE cursor. (SQLCODE -396, SQLSTATE 38505) Por ejemplo, cuando DB2 está en medio de un proceso COMMIT, no puede tolerar nuevo SQL, y cualquier llamada FINAL emitida para una UDF en ese momento será una llamada 255 FINAL. Tenga en cuenta que las UDF que no están definidas para contener un nivel de acceso a SQL nunca recibirán una llamada 255 FINAL, mientras que las UDF que utilizan SQL pueden recibir cualquier tipo de llamada FINAL.

Liberación de recursos

Escriba las rutinas de forma que liberen los recursos que adquieran. Para las funciones de tabla, existen dos lugares naturales para esta liberación: la llamada CLOSE y la llamada FINAL. La llamada CLOSE equilibra cada llamada OPEN y se puede producir varias veces en la ejecución de una sentencia. La llamada FINAL sólo se produce si se especifica FINAL CALL para la UDF, y únicamente se produce una vez por sentencia.

Si puede aplicar un recurso en todas las secuencias de OPEN/FETCH/CLOSE de la UDF, escriba la UDF de forma que adquiera el recurso en la llamada FIRST y lo libere en la llamada FINAL. El área reutilizable es un lugar natural para hacer un seguimiento de este recurso. Para las funciones de tabla, si se especifica FINAL CALL, el área reutilizable sólo se inicializa antes de la llamada FIRST. Si no se especifica FINAL CALL, se reinicializa antes de cada llamada OPEN.

Si un recurso es específico para cada secuencia de OPEN/FETCH/CLOSE, escriba la UDF de forma que libere el recurso en la llamada CLOSE.

Nota: Cuando una función de tabla está en una subconsulta o unión, es muy posible que se produzcan varias apariciones de la secuencia de OPEN/FETCH/CLOSE, según cómo decida organizar la ejecución de la sentencia el Optimizador de DB2.

El *tipo-llamada* toma la forma de un valor INTEGER. DB2 alinea los datos para *tipo-llamada* en función del tipo de datos y del sistema operativo del servidor.

dbinfo

DB2 establece este argumento antes de llamar a la rutina. Sólo está presente si la sentencia CREATE para la rutina especifica la palabra clave DBINFO. El argumento es la estructura `sqludf_dbinfo` definida en el archivo de cabecera `sqludf.h`. Las variables de esta estructura que contienen nombres e identificadores pueden ser más largas que el valor más largo posible en este release de DB2, pero se definen de esta manera a efectos de compatibilidad con releases futuros. Puede utilizar la variable de longitud que complementa cada variable de nombre y de identificador para leer o extraer la porción de la variable que se utilice realmente. La estructura *dbinfo* contiene los elementos siguientes:

1. Longitud del nombre de base de datos (dbnamelen)

   Longitud del *nombre de base de datos* siguiente. Este campo es un entero corto sin signo.

2. Nombre de base de datos (dbname)

   Nombre de la base de datos conectada actualmente. Este campo es un identificador largo de 128 caracteres. El campo *longitud del nombre de base de datos* descrito anteriormente identifica la longitud real de este campo. No contiene un terminador nulo ni ningún relleno.

3. Longitud del ID de autorización de la aplicación (authidlen)

   Longitud del *ID de autorización de la aplicación* siguiente. Este campo es un entero corto sin signo.

4. ID de autorización de la aplicación (authid)

   ID de autorización de ejecución de la aplicación. Este campo es un identificador largo de 128 caracteres. No contiene un terminador nulo ni ningún relleno. El campo *longitud del ID de autorización de la aplicación* descrito anteriormente identifica la longitud real de este campo.

5. Páginas de códigos del entorno (codepg)

   Se trata de una unión de tres estructuras de 48 bytes; una es común para todos los productos de la base de datos de DB2 (cdpg_db2), otra se utiliza en las rutinas escritas para versiones anteriores de la base de datos de DB2 (cdpg_cs) y la última se utiliza en versiones anteriores de DB2 Universal Database para z/OS y OS/390 (cdpg_mvs). Con

miras a la portabilidad, es recomendable que la estructura común, cdpg_db2, se utilice en todas las rutinas.

La estructura cdpg_db2 está formada por una matriz (db2_ccsids_triplet) de tres conjuntos de información de página de códigos que representa los esquemas de codificación posibles de la base de datos del modo siguiente:

a. Esquema de codificación ASCII. Tenga en cuenta que, para mantener la compatibilidad con la versión anterior de la base de datos DB2, si la base de datos es Unicode, la información correspondiente al esquema de codificación Unicode se colocará aquí, además de aparecer en el tercer elemento.
b. Esquema de codificación EBCDIC
c. Esquema de codificación Unicode

Después de la información de los esquemas de codificación, se encuentra el índice de matriz del esquema de codificación para la rutina (db2_encoding_scheme).Cada elemento de la matriz está compuesto por tres campos:

- db2_sbcs. Página de códigos de un solo byte, un entero largo sin signo.
- db2_dbcs. Página de códigos de doble byte, un entero largo sin signo.
- db2_mixed. Página de códigos compuesta (también conocida como página de códigos mixta), un entero largo sin signo.

6. Longitud del nombre de esquema (tbschemalen)

   Longitud del *nombre del esquema* siguiente. Contiene 0 (cero) si no se pasa un nombre de tabla. Este campo es un entero corto sin signo.

7. Nombre del esquema (tbschema)

   Esquema del *nombre de la tabla* siguiente. Este campo es un identificador largo de 128 caracteres. No contiene un terminador nulo ni ningún relleno. El campo *longitud del nombre de esquema* descrito anteriormente identifica la longitud real de este campo.

8. Longitud del nombre de tabla (tbnamelen)

   Longitud del *nombre de la tabla* siguiente. Contiene 0 (cero) si no se pasa un nombre de tabla. Este campo es un entero corto sin signo.

9. Nombre de la tabla (tbname)

   Nombre de la tabla que se está actualizando o insertando. Este campo sólo se establece si la referencia a la rutina se encuentra a la derecha de una cláusula SET de una sentencia UPDATE o si es un elemento de la lista VALUES de una sentencia INSERT. Este campo es un identificador largo de 128 caracteres. No contiene un terminador nulo ni ningún relleno. El campo *longitud del nombre de tabla* descrito anteriormente identifica la longitud real de este campo. El campo *nombre del esquema* descrito anteriormente forma, junto con este campo, el nombre de tabla completamente calificado.

10. Longitud del nombre de columna (colnamelen)

    Longitud del *nombre de columna* siguiente. Contiene un 0 (cero) si no se pasa un nombre de columna. Este campo es un entero corto sin signo.

11. Nombre de la columna (colname)

    Bajo las mismas condiciones exactas que se aplican al nombre de la tabla, este campo contiene el nombre de la columna que se ha de

actualizar o insertar; de lo contrario, es imprevisible. Este campo es un identificador largo de 128 caracteres. No contiene un terminador nulo ni ningún relleno. El campo *longitud del nombre de columna* descrito anteriormente identifica la longitud real de este campo.

12. Número de versión/release (ver_rel)Campo de 8 caracteres que identifica el producto y la versión, el release y el nivel de modificación del mismo, con el formato *pppvvrrm*, donde:
    - *ppp* identifica el producto, del modo siguiente:

      DSN DB2 para z/OS o OS/390

      ARI SQL/DS o DB2 para VM o VSE

      QSQ Base de datos DB2 para System i

      SQL DB2 Database para Linux, UNIX y Windows
    - *vv* es un identificador de versión de dos dígitos.
    - *rr* es un identificador de release de dos dígitos.
    - *m* es un identificador de nivel de modificación de un dígito.
13. Campo reservado (resd0)

    Este campo es para usos futuros.
14. Plataforma (platform)

    Sistema operativo (plataforma) del servidor de aplicaciones, tal como se indica a continuación:

    SQLUDF_PLATFORM_AIX
    AIX

    SQLUDF_PLATFORM_HP
    HP-UX

    SQLUDF_PLATFORM_LINUX
    Linux

    SQLUDF_PLATFORM_MVS
    OS/390

    SQLUDF_PLATFORM_NT
    Windows 2000, Windows XP

    SQLUDF_PLATFORM_SUN
    Sistema operativo Solaris

    SQLUDF_PLATFORM_WINDOWS95
    Windows 95, Windows 98, Windows Me

    SQLUDF_PLATFORM_UNKNOWN
    Plataforma o sistema operativo desconocido

    Para informarse sobre sistemas operativos adicionales no incluidos en la lista anterior, consulte el contenido del archivo `sqludf.h`.
15. Número de entradas de la lista de columnas de la función de tabla (numtfcol)

    Número de entradas distintas de cero que existen en la lista de columnas de la función de tabla especificada en el campo *lista de columnas de la función de tabla* siguiente.
16. Campo reservado (resd1)

    Este campo es para usos futuros.

17. ID de rutina del procedimiento almacenado que ha invocado la rutina actual (procid)

    El ID de rutina del procedimiento almacenado coincide con la columna ROUTINEID de SYSCAT.ROUTINES, que se puede utilizar para recuperar el nombre del procedimiento almacenado que realiza la invocación. Este campo es un entero con signo de 32 bits.

18. Campo reservado (resd2)

    Este campo es para usos futuros.

19. Lista de columnas de la función de tabla (tfcolumn)

    Si ésta es una función de tabla, este campo es un puntero a una matriz de enteros cortos que DB2 asigna dinámicamente. Si se trata de cualquier otro tipo de rutina, este puntero es nulo.

    Sólo se utiliza este campo para las funciones de tabla. Sólo son de interés las *n* primeras entradas, donde *n* se especifica en el campo *número de entradas de la lista de columnas de la función de tabla*, `numtfcol`. *n* puede equivaler a 0, y *n* es inferior o igual al número de columnas de resultado definidas para la función en la cláusula RETURNS TABLE(...) de la sentencia CREATE FUNCTION. Los valores corresponden a los números ordinales de las columnas que esta sentencia necesita de la función de tabla. Un valor de '1' significa la primera columna de resultado definida, '2' significa la segunda columna de resultado definida, y así sucesivamente, y los valores pueden seguir cualquier orden. Tenga en cuenta que *n* puede ser igual a cero, es decir, que la variable `numtfcol` puede ser cero, para una sentencia parecida a `SELECT COUNT(*) FROM TABLE(TF(...)) AS QQ`, en que la consulta no necesita ningún valor de columna real.

    Esta matriz representa una oportunidad para realizar una optimización. La UDF no tiene necesidad de devolver todos los valores de todas las columnas de resultado de la función de tabla, únicamente de aquéllas que sean necesarias en el contexto en concreto, y éstas son las columnas identificadas (por número) en la matriz. Dado que esta optimización puede complicar la lógica de la UDF para lograr beneficios en el rendimiento, la UDF puede elegir devolver todas las columnas definidas.

20. Identificador exclusivo de la aplicación (appl_id)

    Este campo es un puntero a una serie C terminada en nulo que identifica de forma exclusiva la conexión de la aplicación con DB2. DB2 lo genera durante la conexión.

    La serie tiene una longitud máxima de 32 caracteres y su formato exacto depende del tipo de conexión establecida entre el cliente y DB2. Generalmente, toma la forma siguiente:

    ```
    x.y.ts
    ```

    donde *x* e *y* varían según el tipo de conexión, pero *ts* es una indicación de la hora de 12 caracteres con formato AAMMDDHHMMSS, que DB2 ajusta potencialmente para asegurar su exclusividad.

    ```
    Ejemplo:  *LOCAL.db2inst.980707130144
    ```

21. Campo reservado (resd3)

    Este campo es para usos futuros.

Variables gráficas del lenguaje principal en rutinas C y C++

Generalmente, cualquier rutina escrita en C o C++ que reciba o devuelva datos gráficos a través de su entrada o salida de parámetros, se debe precompilar con la opción WCHARTYPE NOCONVERT. Esto es debido a que se considera que los datos gráficos que se pasan mediante dichos parámetros están en formato DBCS, en lugar del formato de código de procesos `wchar_t`. La utilización de NOCONVERT significa que los datos gráficos manipulados en sentencias de SQL de la rutina también estarán en formato DBCS, coincidiendo así con el formato de los datos de parámetros.

Con WCHARTYPE NOCONVERT, no se produce ninguna conversión del código de caracteres entre la variable gráfica del lenguaje principal y el gestor de bases de datos. Los datos de una variable gráfica del lenguaje principal se envían al gestor de bases de datos y se reciben del mismo como caracteres DBCS inalterados. Si no utiliza WCHARTYPE NOCONVERT, también podrá manipular datos gráficos en formato `wchar_t` en una rutina; sin embargo, deberá realizar manualmente las conversiones de entrada y salida.

Se puede utilizar CONVERT en las rutinas FENCED, lo que afectará a los datos gráficos de las sentencias de SQL incluidas en la rutina, pero no a los datos pasados a través de los parámetros de la rutina. Las rutinas NOT FENCED se deben crear utilizando la opción NOCONVERT.

En resumen, los datos gráficos pasados a una rutina o devueltos por ésta mediante sus parámetros de entrada o salida están en formato DBCS, independientemente de cómo se haya precompilado con la opción WCHARTYPE.

Decoración de tipos de C++

Los nombres de las funciones de C++ se pueden sobrecargar. Pueden coexistir dos funciones de C++ con el mismo nombre si tienen argumentos distintos, por ejemplo:

```
int func( int i )
```

e

```
int func( char c )
```

Por omisión, los compiladores C++ decoran con tipos o 'despedazan' los nombres de función. Esto significa que se añaden nombres de tipo de argumento a sus nombres de función correspondientes para resolverlos, como por ejemplo en `func__Fi` y `func__Fc` para los dos ejemplos anteriores. Los nombres despedazados serán distintos en cada sistema operativo, por lo que no se puede transportar el código que utiliza explícitamente un nombre despedazado.

En los sistemas operativos Windows, el nombre de función decorado con tipos se puede determinar a partir del archivo `.obj` (objeto).

Con el compilador Microsoft Visual C++ en Windows, puede utilizar el mandato `dumpbin` para determinar el nombre de función decorado con tipos a partir del archivo `.obj` (objeto), del modo siguiente:

```
dumpbin /symbols myprog.obj
```

donde `myprog.obj` es el archivo de objeto del programa.

En los sistemas operativos UNIX, el nombre de función decorado con tipos se puede determinar a partir del archivo `.o` (objeto), o de la biblioteca compartida, utilizando el mandato `nm`. Este mandato puede producir una salida considerable, por lo que se le aconseja que conduzca la salida a través de `grep` para buscar la línea correcta, del modo siguiente:

```
nm myprog.o | grep myfunc
```

donde `myprog.o` es el archivo de objeto del programa y `myfunc` es la función en el archivo fuente del programa.

La salida producida por todos estos mandatos incluye una línea con el nombre de función despedazado. Por ejemplo, en UNIX, dicha línea es parecida a la siguiente:

```
myfunc__FPlT1PsT3PcN35|     3792|unamex|          | ...
```

Una vez obtenido el nombre de función despedazado de uno de los mandatos anteriores, lo podrá utilizar en el mandato apropiado. Esto se muestra más adelante en este apartado al utilizar el nombre de función despedazado que se ha obtenido en el ejemplo de UNIX anterior. Un nombre de función despedazado obtenido en Windows se utilizaría de la misma manera.

Cuando se registra una rutina con la sentencia CREATE, la cláusula EXTERNAL NAME debe especificar el nombre de función despedazado. Por ejemplo:

```
CREATE FUNCTION myfunco(...) RETURNS...
       ...
       EXTERNAL NAME '/whatever/path/myprog!myfunc__FPlT1PsT3PcN35'
       ...
```

Si la biblioteca de rutinas no contiene nombres de funciones de C++ sobrecargados, existe la opción de utilizar `extern "C"` para hacer que el compilador no decore con tipos los nombres de función. (Tenga en cuenta que siempre puede sobrecargar los nombres de función de SQL asignados a las UDF, ya que DB2 resuelve qué función de biblioteca debe invocar basándose en el nombre y los parámetros que toma.)

```
#include <string.h>
#include <stdlib.h>
#include "sqludf.h"

/*------------------------------------------------------------------------*/
/* función fold: salida = la serie de entrada se dobla en el punto     */
/*                        indicado por el segundo argumento.           */
/*         entradas: CLOB,               serie de entrada              */
/*                   LONG                posición por la que doblar    */
/*         salida:   CLOB                serie doblada                 */
/*------------------------------------------------------------------------*/
extern "C" void fold(
      SQLUDF_CLOB       *in1,                /* entrada CLOB a doblar */
   ...
   ...
}
/* fin de UDF: fold */

/*------------------------------------------------------------------------*/
/* función find_vowel:                                                    */
/*             devuelve la posición de la primera vocal                   */
/*             devuelve un error si no hay ninguna vocal                  */
/*             definida como NOT NULL CALL                                */
/*         entradas: VARCHAR(500)                                         */
/*         salida:   INTEGER                                              */
/*------------------------------------------------------------------------*/
extern "C" void findvwl(
      SQLUDF_VARCHAR    *in,                      /* entrada smallint */
   ...
   ...
}
/* fin de UDF: findvwl */
```

En este ejemplo, el compilador no decora con tipos las UDF `fold` y `findvwl`, que se deben registrar en la sentencia CREATE FUNCTION utilizando sus nombres llanos. De forma similar, si un método o procedimiento almacenado C++ se codifica con `extern "C"`, se utilizará su nombre de función no decorado en la sentencia CREATE.

Devolución de conjuntos de resultados procedentes de procedimientos C y C++

Puede desarrollar procedimientos C y C++ que devuelvan conjuntos de resultados a una rutina o aplicación llamadora implementada mediante una API que permita recuperar conjuntos de resultados de procedimientos. La mayoría de las API da soporte a la recuperación de conjuntos de resultados de procedimientos, pero no en el caso de SQL incorporado.

La representación C y C++ de un conjunto de resultados es un cursor SQL. Cualquier cursor SQL que se haya declarado y abierto, pero no explícitamente cerrado dentro de un procedimiento, antes del retorno del procedimiento se puede devolver al llamador. El orden en que se devuelven los conjuntos de resultados al llamador coincide con el orden en que los objetos cursor se abren dentro de la rutina. No se necesitan parámetros adicionales en la sentencia CREATE PROCEDURE ni en la implementación del procedimiento para devolver un conjunto de resultados.

Requisitos previos

Un conocimiento general de cómo se crean las rutinas C y C++ le ayudará a seguir los pasos del procedimiento que figura más abajo para devolver resultados de un procedimiento C o C++.

Los cursores declarados en procedimientos de SQL incorporado en C o C++ no son cursores desplazables.

Procedimiento

Para devolver un conjunto de resultados de un procedimiento C o C++:

1. En la sentencia CREATE PROCEDURE del procedimiento C o C++, debe especificar, junto con cualquier otra cláusula pertinente, la cláusula DYNAMIC RESULT SETS con un valor igual al número máximo de conjuntos de resultados que debe devolver el procedimiento.
2. Los marcadores de parámetros no son necesarios en la declaración de un procedimiento para un conjunto de resultados que se va a devolver al llamador.
3. En la implementación de procedimiento C o C++ de la rutina, declare un cursor mediante la sentencia DECLARE CURSOR en la sección de declaraciones en las que se declaran las variables del lenguaje principal. La declaración del cursor asocia un SQL al cursor.
4. Dentro del código de la rutina C o C++, abra el cursor ejecutando la sentencia OPEN. Con ello se ejecuta la consulta especificada en la sentencia DECLARE CURSOR, y el resultado de la consulta se asocia al cursor.
5. Opcional: capte filas del conjunto de resultados asociado al cursor mediante la sentencia FETCH.
6. No ejecute la sentencia CLOSE que sirve para cerrar el cursor en ningún momento anterior al retorno del procedimiento al llamador. El cursor abierto se devolverá como conjunto de resultados al llamador cuando el procedimiento retorne.

   Cuando se deja abierto más de un cursor después del retorno de un procedimiento, los conjuntos de resultados asociados a los cursores se devuelven al llamador en el orden en que se abrieron. Con el procedimiento no se pueden devolver más del número máximo de conjuntos de resultados especificados por el valor de la cláusula DYNAMIC RESULT SETS. Si el número de cursores que se han dejado abiertos en la implementación del procedimiento es mayor que el valor especificado por la cláusula DYNAMIC RESULT SETS, los conjuntos de resultados en exceso no se devuelven. Cuando se da esta situación, DB2 no indica ningún mensaje de error o aviso.

Una vez que la creación del procedimiento C o C++ se ha completado satisfactoriamente, ya se puede invocar el procedimiento con la sentencia CALL desde el procesador de línea de mandatos (CLP) de DB2 o desde una ventana de mandatos de DB2 para verificar que los conjuntos de resultados se han devuelto satisfactoriamente al llamador.

Si desea obtener información sobre la llamada a procedimientos y otros tipos de rutinas:

- Invocación de la rutina

Creación de rutinas C y C++

Los procedimientos y funciones que hacen referencia a una biblioteca C o C++ se crean de forma similar a las rutinas externas con otras implementaciones. Esta tarea comprende unos pasos, incluida la formulación de la sentencia CREATE para la rutina, la codificación de la implementación de la rutina, la precompilación, la compilación y enlace de código, y el despliegue de código fuente.

Elegirá implementar una rutina C o C++ si:

- Desea encapsular lógica compleja en una rutina que acceda a la base de datos o que realice una acción fuera de la base de datos.
- Necesita que la lógica encapsulada se invoque desde cualquiera de estos elementos: diversas aplicaciones, el CLP, otra rutina (procedimiento, función (UDF) o método) o un activador.
- Se siente más cómodo codificando esta lógica con lenguaje de programación de SQL incorporado como C o C++.

Requisitos previos

- Conocimiento de la implementación de rutinas C y C++. Para obtener información sobre las rutinas C y C++ en general, consulte:
  - "Rutinas C y C++" en la página 216
- El Cliente DB2, que incluye el soporte al desarrollo de aplicaciones debe estar instalado en el sistema cliente.
- El servidor de bases de datos debe ejecutar un sistema operativo que dé soporte a un compilador C o C++ soportado de DB2 para el desarrollo de rutinas.
- Los compiladores necesarios deben estar instalados en el servidor de bases de datos.
- Autorización para ejecutar la sentencia CREATE correspondiente a la rutina externa. Si desea saber cuáles son los privilegios necesarios para ejecutar la sentencia CREATE PROCEDURE o la sentencia CREATE FUNCTION, consulte la documentación de la sentencia.

Procedimiento

1. Codifique la lógica de la rutina en el lenguaje de programación elegido: C o C++.
   - Para obtener información general sobre rutinas C y C++ y características de las rutinas C y C++, consulte los temas a los que se hace referencia en la sección Requisitos previos.
   - Incluya los archivos de cabecera de C o C++ necesarios para funciones adicionales de C, así como los archivos de cabecera de C o C++ de DB2 necesarios para el tipo de datos de SQL y el soporte de ejecución de SQL. Incluya los siguientes archivos de cabecera: sqludf.h, sql.h, sqlda.h, sqlca.h y memory.h.
   - Debe implementarse una signatura de parámetro de rutina utilizando uno de los estilos de parámetros soportados. Se recomienda utilizar el estilo de parámetro SQL para todas las rutinas C y C++. Las áreas reutilizables y las estructuras dbinfo se pasan a las rutinas C y C++ como parámetros. Si desea más información sobre las signaturas de parámetros y las implementaciones de parámetros, consulte:
     - "Parámetros en rutinas C y C++" en la página 220
     - "Estilo de parámetro de SQL en procedimientos C y C++" en la página 222

– "Estilo de parámetro de SQL en funciones C y C++" en la página 225

- Declare variables del lenguaje principal y marcas de parámetros de la misma manera que se hace para las aplicaciones C y C++ de SQL incorporado. Utilice correctamente los tipos de datos que se correlacionan con tipos de datos de SQL de DB2. Para obtener más información sobre la correlación de tipos de datos entre DB2 y C o C++, consulte:
  – "Tipos de datos de SQL soportados en rutinas C y C++" en la página 233
- Incluya la lógica de la rutina. La lógica de la rutina puede constar de cualquier código soportado en el lenguaje de programación C o C++. También puede incluir la ejecución de sentencias de SQL incorporado, que se implementan de la misma manera que para las aplicaciones de SQL incorporado. Para obtener más información sobre la ejecución de sentencias de SQL en SQL incorporado, consulte:
  – "Ejecución de sentencias de SQL en aplicaciones de SQL incorporado" en *Developing Embedded SQL Applications*
- Si la rutina es un procedimiento y desea devolver un conjunto de resultados al llamador de la rutina, no es necesario ningún parámetro para el conjunto de resultados. Si desea más información sobre la devolución de conjuntos de resultados desde rutinas:
  – "Devolución de conjuntos de resultados procedentes de procedimientos C y C++" en la página 259
- Establezca un valor de retorno de rutina al final de la rutina.

2. Construya el código para generar un archivo de biblioteca. Para obtener información sobre cómo crear rutinas C y C++ de SQL incorporado, consulte:
   - "Creación del código de rutinas C y C++" en la página 263
3. Copie la biblioteca del directorio *function* de la base de datos de DB2. Es recomendable almacenar las bibliotecas asociadas con rutinas de DB2 en el directorio de función. Para conocer más acerca del directorio de función, consulte la cláusula EXTERNAL de una de las sentencias siguientes: CREATE PROCEDURE o CREATE FUNCTION.

   Puede copiar la biblioteca en otro directorio del servidor, pero para invocar la rutina satisfactoriamente, debe anotar el nombre de la vía de acceso totalmente calificada de la biblioteca, ya que lo necesitará en el paso siguiente.
4. Ejecute de forma dinámica o estática la sentencia CREATE de lenguaje SQL correspondiente para el tipo de rutina: CREATE PROCEDURE o CREATE FUNCTION.
   - Especifique la cláusula `LANGUAGE` con el valor: C
   - Especifique la cláusula `PARAMETER STYLE` con el nombre del estilo de parámetro soportado que se ha implementado en el código de la rutina. Se recomienda utilizar PARAMETER STYLE SQL.
   - Especifique la cláusula `EXTERNAL` con el nombre de la biblioteca que se ha de asociar con la rutina utilizando uno de los valores siguientes:
     – el nombre de vía de acceso completamente calificado de la biblioteca de rutina
     – el nombre de vía de acceso relativo de la biblioteca de rutina en relación con el directorio de función.

     Por omisión, DB2 buscará la biblioteca en el directorio de función, a menos que se especifique un nombre de vía de acceso completamente calificado o relativo para la biblioteca en la cláusula EXTERNAL.

- Especifique DYNAMIC RESULT SETS con un valor numérico si la rutina es un procedimiento y ha de devolver uno o varios conjuntos de resultados al llamador.
- Especifique otros valores de cláusula no por omisión en la sentencia CREATE a utilizar para caracterizar la rutina.

Para invocar la rutina C o C++, consulte el Capítulo 11, "Invocación de rutinas", en la página 355.

Creación del código de rutinas C y C++

Una vez se ha escrito el código de implementación de una rutina C o C++ de SQL incorporado, se debe incorporar en una biblioteca y desplegar para que se pueda invocar la rutina. Aunque los pasos a seguir para crear rutinas C y C++ de SQL incorporado son parecidos a los necesarios para crear aplicaciones C y C++ de SQL incorporado, hay algunas diferencias. Los mismos pasos se pueden seguir si no hay sentencias de SQL incorporado dentro de las rutinas; el procedimiento será más rápido y sencillo.

Hay dos formas de crear rutinas C y C++:

- Mediante scripts de creación de ejemplo de DB2 (UNIX) o archivos de proceso por lotes de creación (Windows)
- Especificando mandatos del compilador de DB2 y C o C++ desde una ventana de mandatos de DB2

Los scripts de creación de ejemplo de DB2 y los archivos de proceso por lotes correspondientes a rutinas están diseñados para crear rutinas de ejemplo de DB2 (procedimientos y funciones definidas por el usuario), así como rutinas creadas por el usuario para un determinado sistema operativo utilizando los compiladores soportados por omisión.
Hay un conjunto separado de scripts de creación de ejemplo de DB2 y de archivos de proceso por lotes para C y C++. En general, es más fácil crear rutinas de SQL incorporado utilizando los scripts de creación o los archivos de proceso por lotes, que se pueden modificar fácilmente si hace falta; sin embargo, suele resultar útil saber cómo crear rutinas desde ventanas de mandatos de DB2.

Creación de código de rutinas C y C++ mediante scripts bldrtn de ejemplo

La creación de código fuente de rutinas C y C++ es una subtarea de la creación de rutinas C y C++. Esta tarea se puede realizar de forma rápida y fácil mediante los scripts de creación de ejemplo (UNIX) y los archivos de proceso por lotes (Windows) de DB2. Los scripts de creación de ejemplo se pueden utilizar para el código fuente con o sin sentencias de SQL incorporado. Los scripts de creación se ocupan de la precompilación, compilación y enlace del código fuente C y C++, que de otro modo se tiene que hacer con pasos individuales desde la línea de mandatos. También se ocupan de vincular cualquier paquete a la base de datos especificada.

Los scripts de creación de ejemplo para crear rutinas C y C++ se denominan `bldrtn`. Se encuentran en directorios de DB2 junto con los programas de ejemplo que se pueden crear con ellos, en las siguientes ubicaciones:

- Para C: `sqllib/samples/c/`
- Para C++: `sqllib/samples/cpp/`

El script `bldrtn` se puede utilizar para crear un archivo de código fuente que contenga implementaciones de procedimientos y funciones. El script hace lo siguiente:

- Establece una conexión con una base de datos especificada por el usuario
- Precompila el archivo de código fuente especificado por el usuario
- Vincula el paquete con la base de datos actual
- Compila y enlaza el código fuente para generar una biblioteca compartida
- Copia la biblioteca compartida en el directorio de función de DB2 en el servidor de bases de datos

Los scripts `bldrtn` aceptan dos argumentos:

- El nombre de un archivo de código fuente sin extensión de archivo
- El nombre de una base de datos con la que se establecerá una conexión

El parámetro correspondiente a la base de datos es opcional. Si no se proporciona un nombre de base de datos, el programa utiliza la base de datos por omisión sample. Puesto que las rutinas se tienen que crear en la misma instancia en la que reside la base de datos, no se necesitan argumentos para ID de usuario y contraseña.

Requisitos previos

- Archivo de código fuente que contenga una o más implementaciones de rutinas.
- El nombre de la base de datos dentro de la instancia de DB2 actual en la que se van a crear las rutinas.

Para crear un archivo de código fuente que contenga una o más implementaciones de código de rutina, siga los pasos siguientes.

1. Abra una ventana de mandatos de DB2.
2. Copie el archivo de código fuente en el mismo directorio que el script `bldrtn`.
3. Si las rutinas se van a crear en la base de datos sample, especifique el nombre del script de creación seguido del nombre del archivo de código fuente sin la extensión de archivo .sqc o .sqC.

   ```
   bldrtn <nombre-archivo>
   ```

   Si las rutinas se van a crear en otra base de datos, especifique el nombre del script de creación, el nombre del archivo de código fuente sin extensión de archivo y el nombre de la base de datos:

   ```
   bldrtn <nombre-archivo> <nombre-basedatos>
   ```

   El script precompila, compila y enlaza el código fuente y genera una biblioteca compartida. A continuación, el script copia la biblioteca compartida en el directorio de función en el servidor de bases de datos.
4. Si no es la primera vez que se crea el archivo de código fuente que contiene las implementaciones de rutinas, detenga y vuelva a iniciar la base de datos para asegurarse de que DB2 utiliza la nueva versión de la biblioteca compartida. Puede hacerlo especificando `db2stop` seguido de db2start en la línea de mandatos.

Cuando haya creado satisfactoriamente la biblioteca compartida de rutinas y la haya desplegado en el directorio de función en el servidor de bases de datos, debe completar los pasos asociados a la tarea de crear rutinas C y C++. Una vez completada la creación de rutinas, podrá invocar las rutinas.

Creación de rutinas en C o C++ mediante el script de creación de ejemplo (UNIX)

DB2 proporciona scripts de creación para compilar y enlazar programas en C y C++. Se encuentran en el directorio `sqllib/samples/c` para rutinas en C y en el directorio `sqllib/samples/cpp` para rutinas en C++, junto con programas de ejemplo que se pueden crear con estos archivos.

El script `bldrtn` contiene los mandatos para crear rutinas (procedimientos almacenados y funciones definidas por el usuario). El script compila la rutina y crea una biblioteca compartida que puede ser cargada por el gestor de bases de datos y llamada por una aplicación cliente.

El primer parámetro, `$1`, especifica el nombre del archivo fuente. El segundo parámetro, `$2`, especifica el nombre de la base de datos a la que desea conectarse.

El parámetro correspondiente a la base de datos es opcional. Si no se proporciona un nombre de base de datos, el programa utiliza la base de datos por omisión `sample`. Debido a que el procedimiento almacenado se debe crear en la misma instancia donde reside la base de datos, no hay parámetros para el ID de usuario ni la contraseña.

Los ejemplos siguientes muestran cómo crear bibliotecas compartidas de rutinas mediante:

- procedimientos almacenados
- funciones definidas por el usuario (UDF) de SQL no incorporado
- funciones definidas por el usuario (UDF) de SQL incorporado

Biblioteca compartida de procedimientos almacenados

Puede crear el programa de ejemplo `spserver` a partir del archivo fuente `spserver.sqc` para C y `spserver.sqC` para C++:

1. Si conecta con la base de datos `sample`, especifique el nombre del script de creación y el nombre del programa:

   ```
   bldrtn spserver
   ```

   Si conecta con otra base de datos, especifique también el nombre de la base de datos:

   ```
   bldrtn spserver basedatos
   ```

   El script copia la biblioteca compartida en el directorio `sqllib/function` del servidor.
2. A continuación, catalogue las rutinas ejecutando el script `spcat` en el servidor:

   ```
   spcat
   ```

   Este script conecta con la base de datos "sample", descataloga mediante `spdrop.db2` las rutinas que se hubieran catalogado previamente, luego las cataloga llamando a `spcreate.db2`, y finalmente desconecta de la base de datos. Puede también ejecutar los scripts `spdrop.db2` y `spcreate.db2` por separado.
3. A continuación, si no es la primera vez que se crea el procedimiento almacenado, detenga y reinicie la base de datos para asegurarse que se reconoce la nueva versión de la biblioteca compartida. Lo puede hacer entrando `db2stop` seguido de `db2start` en la línea de mandatos.

Una vez creada la biblioteca compartida, spserver, puede crear la aplicación cliente, spclient, la cual accede a la biblioteca compartida.

Puede crear spclient utilizando el archivo de script, bldapp.

Para invocar los procedimientos almacenados en la biblioteca compartida, ejecute la aplicación cliente de ejemplo, especificando lo siguiente: spclient *basedatos IDusuario contraseña*

donde

basedatos
Es el nombre de la base de datos a la que desea conectarse. El nombre podría ser sample, o su alias, u otro nombre de base de datos.

IDusuario
Es un ID de usuario válido.

contraseña
Es una contraseña válida correspondiente al ID de usuario.

La aplicación cliente accede a la biblioteca compartida, spserver, la cual ejecuta diversas funciones de procedimiento almacenado contenidas en la base de datos del servidor. Los datos resultantes se devuelven a la aplicación cliente.

Biblioteca compartida de UDF de SQL incorporado

Puede crear el programa de función definida por el usuario de SQL incorporado, udfemsrv, a partir del archivo fuente udfemsrv.sqc para C y udfemsrv.sqC para C++. Si conecta con la base de datos sample, especifique el nombre del script de creación y el nombre del programa:

```
bldrtn udfemsrv
```

Si conecta con otra base de datos, especifique también el nombre de la base de datos:

```
bldrtn udfemsrv basedatos
```

El script copia la UDF en el directorio sqllib/function.

Una vez creado el programa udfemsrv, puede crear la aplicación cliente, udfemcli, que llama al programa. Puede crear el programa cliente udfemcli a partir del archivo fuente udfemcli.sqc, contenido en sqllib/samples/c, utilizando el script bldapp.

Para invocar las UDF de la biblioteca compartida, ejecute la aplicación cliente, especificando lo siguiente: udfemcli *basedatos IDusuario contraseña*

donde

basedatos
Es el nombre de la base de datos a la que desea conectarse. El nombre podría ser sample, o su alias, u otro nombre de base de datos.

IDusuario
Es un ID de usuario válido.

contraseña
Es una contraseña válida correspondiente al ID de usuario.

La aplicación cliente accede a la biblioteca compartida, udfemsrv, y ejecuta las funciones definidas por el usuario contenidas en la base de datos del servidor. Los datos resultantes se devuelven a la aplicación cliente.

Nota: La variable de entorno de sistema operativo LD_LIBRARY_PATH se ignora cuando se crean procedimientos de SQL incorporado.

Creación de rutinas 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 y 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 bldrtn.bat contiene los mandatos para crear rutinas (procedimientos almacenados y funciones definidas por el usuario) de SQL incorporado. El archivo de proceso por lotes crea una DLL en el servidor. Utiliza dos parámetros como entrada, que están representados dentro del archivo de proceso por lotes por las variables %1 y %2.

El primer parámetro, %1, especifica el nombre del archivo fuente. El archivo de proceso por lotes utiliza el nombre del archivo fuente para el nombre de la DLL El segundo parámetro, %2, especifica el nombre de la base de datos a la que desea conectarse. Debido a que la DLL se debe crear en la misma instancia donde reside la base de datos, no hay parámetros para el ID de usuario ni la contraseña.

Sólo es obligatorio el primer parámetro, el nombre del archivo fuente. El nombre de la base de datos es opcional. Si no se proporciona un nombre de base de datos, el programa utiliza la base de datos por omisión sample.

Los ejemplos siguientes muestran cómo crear las DLL de rutinas mediante:

- procedimientos almacenados
- funciones definidas por el usuario (UDF) de SQL no incorporado
- funciones definidas por el usuario (UDF) de SQL incorporado

DLL de procedimiento almacenado

Para crear la DLL spserver a partir del archivo fuente C, spserver.sqc, o a partir del archivo fuente C++, spserver.sqx:

1. Escriba el nombre del archivo de proceso por lotes y el nombre del programa:

   ```
   bldrtn spserver
   ```

   Si conecta con otra base de datos, especifique también el nombre de la base de datos:

   ```
   bldrtn spserver basedatos
   ```

   El archivo de proceso por lotes utiliza el archivo de definición de módulos, spserver.def, que reside en el mismo directorio que los programas de ejemplo, para crear la DLL. El archivo de proceso por lotes copia la DLL, spserver.dll, en el directorio sqllib\function del servidor.
2. A continuación, catalogue las rutinas ejecutando el script spcat en el servidor:

   ```
   spcat
   ```

   Este script conecta con la base de datos "sample", descataloga mediante spdrop.db2 las rutinas que se hubieran catalogado previamente, luego las cataloga llamando a spcreate.db2, y finalmente desconecta de la base de datos. Puede también ejecutar los scripts spdrop.db2 y spcreate.db2 por separado.

3. A continuación, detenga y rearranque la base de datos para que se reconozca la nueva DLL. Si es necesario, establezca la modalidad de archivo para la DLL a fin de que la instancia de DB2 pueda acceder a la DLL.

Una vez creada la DLL, `spserver`, puede crear la aplicación cliente `spclient`, la cual accede a la DLL.

Puede crear `spclient` utilizando el archivo de proceso por lotes, `bldapp.bat`.

Para invocar la DLL, ejecute la aplicación cliente de ejemplo, especificando lo siguiente:

`spclient` *basedatos IDusuario contraseña*

donde

basedatos
Es el nombre de la base de datos a la que desea conectarse. El nombre podría ser `sample`, o su alias, u otro nombre de base de datos.

IDusuario
Es un ID de usuario válido.

contraseña
Es una contraseña válida correspondiente al ID de usuario.

La aplicación cliente accede a la DLL, `spserver`, la cual ejecuta varias rutinas contenidas en la base de datos del servidor. Los datos resultantes se devuelven a la aplicación cliente.

DLL de funciones definidas por el usuario de SQL no incorporado

Para crear la función definida por el usuario `udfsrv` a partir del archivo fuente `udfsrv.c`, especifique:

```
bldrtn udfsrv
```

El archivo de proceso por lotes utiliza el archivo de definición de módulos, `udfsrv.def`, que reside en el mismo directorio que los programas de ejemplo, para crear la DLL de función definida por el usuario. El archivo de proceso por lotes copia la DLL de función definida por el usuario, `udfsrv.dll`, en el directorio `sqllib\function` del servidor.

Una vez creado el programa `udfsrv`, puede crear la aplicación cliente, `udfcli`, la cual llama al programa. Se proporcionan versiones en C y C++ de este programa para la CLI de DB2 y de SQL incorporado.

Puede crear el programa `udfcli` de la CLI de DB2 a partir del archivo fuente `udfcli.c` situado en `sqllib\samples\cli` utilizando el archivo de proceso por lotes `bldapp`.

Puede crear el programa en C `udfcli` de SQL incorporado a partir del archivo fuente `udfcli.sqc`, situado en `sqllib\samples\c`, utilizando el archivo de proceso por lotes `bldapp`.

Puede crear el programa en C++ `udfcli` de SQL incorporado a partir del archivo fuente `udfcli.sqx`, situado en `sqllib\samples\cpp`, utilizando el archivo de proceso por lotes `bldapp`.

Para ejecutar la UDF (función definida por el usuario), especifique:

```
udfcli
```

La aplicación solicitante llama a la función `ScalarUDF` de la DLL `udfsrv`.

DLL de UDF de SQL incorporado

Para crear la biblioteca de función definida por el usuario de SQL incorporado `udfemsrv`, a partir del archivo fuente C `udfemsrv.sqc` de `sqllib\samples\c`, o a partir del archivo C++ `udfemsrv.sqx` de `sqllib\samples\cpp`, especifique:

```
bldrtn udfemsrv
```

Si conecta con otra base de datos, especifique también el nombre de la base de datos:

```
bldrtn udfemsrv basedatos
```

El archivo de proceso por lotes utiliza el archivo de definición de módulos, `udfemsrv.def`, que reside en el mismo directorio que los programas de ejemplo, para crear la DLL de función definida por el usuario. El archivo de proceso por lotes copia la DLL de función definida por el usuario, `udfemsrv.dll`, en el directorio `sqllib\function` del servidor.

Una vez creado el programa `udfemsrv`, puede crear la aplicación cliente, `udfemcli`, que llama al programa. Puede crear `udfemcli` a partir del archivo fuente C `udfemcli.sqc` de `sqllib\samples\c`, o a partir del archivo fuente C++ `udfemcli.sqx` de `sqllib\samples\cpp`, utilizando el archivo de proceso por lotes `bldapp`.

Para ejecutar la UDF (función definida por el usuario), especifique:

```
udfemcli
```

La aplicación solicitante llama a las UDF contenidas en la DLL `udfemsrv`.

Creación de códigos de rutinas C y C++ desde ventanas de mandatos de DB2

La creación de código fuente de rutinas C y C++ es una subtarea de la creación de rutinas C y C++. Esta tarea se puede realizar de forma manual desde la línea de mandatos. Se puede seguir el mismo procedimiento independientemente de si hay o no sentencias de SQL incorporado dentro del código de rutinas C o C++. Los pasos de la tarea incluyen precompilación, compilación y enlace de código fuente C y C++ que contiene implementaciones de rutinas, vinculación del paquete generado (si había sentencias de SQL incorporado) y despliegue de la biblioteca compartida. Puede realizar esta tarea desde una ventana de mandatos de DB2 como parte de la prueba del uso de un precompilador, compilador u opción de vinculación, si desea diferir la vinculación de los paquetes de la rutina hasta más adelante o si está desarrollando scripts de creación personalizados.

Como alternativa, puede utilizar los scripts de creación de ejemplo de DB2 para simplificar esta tarea. Consulte: Creación de código de rutinas C y C++ de SQL incorporado mediante scripts de creación de ejemplo.

Requisitos previos

- El archivo de código fuente que contenga una o más implantaciones de rutinas C o C++ de SQL incorporado.

- El nombre de la base de datos dentro de la instancia de DB2 actual en la que se van a crear las rutinas.
- Las opciones de compilación y enlace específicas del sistema operativo necesarias para crear rutinas C y C++. Consulte los temas a los que se hace referencia en los enlaces relacionados al final de este tema.

Para crear un archivo de código fuente que contenga una o más implementaciones de código de rutina, siga los pasos siguientes. A continuación se describe un ejemplo que muestra cada uno de los pasos:

1. Abra una ventana de mandatos de DB2.
2. Navegue hasta el directorio que contiene el archivo de código fuente.
3. Establezca una conexión con la base de datos en la que se van a crear las rutinas.
4. Precompile el archivo de código fuente.
5. Enlace el paquete que se ha generado con la base de datos.
6. Compile el archivo de código fuente.
7. Enlace el archivo de código fuente para generar una biblioteca compartida. Para ello hay que utilizar algunas opciones de compilación y enlace específicas de DB2 correspondientes a compilador que se utiliza.
8. Copie la biblioteca compartida en el directorio de función de DB2 en el servidor de bases de datos.
9. Si no es la primera vez que se crea el archivo de código fuente que contiene las implementaciones de rutinas, detenga y vuelva a iniciar la base de datos para asegurarse de que DB2 utiliza la nueva versión de la biblioteca compartida. Puede hacerlo emitiendo el mandato db2stop seguido del mandato db2start.

Cuando haya creado y desplegado satisfactoriamente la biblioteca de rutinas, debe completar los pasos asociados a la tarea de crear rutinas C y C++. La creación de rutinas C y C++ incluye un paso para ejecutar la sentencia CREATE para cada rutina implementada en el archivo de código fuente. Este paso también de debe completar para poder invocar las rutinas.

Ejemplo

El siguiente ejemplo muestra la recreación de un archivo de código fuente C++ de SQL incorporado denominado myfile.sqC que contiene implementaciones de rutinas. Las rutinas se crean en un sistema operativo AIX utilizando el compilador IBM VisualAge C++ soportado por omisión para generar una biblioteca de rutinas de 32 bits.

1. Abra una ventana de mandatos de DB2.
2. Navegue hasta el directorio que contiene el archivo de código fuente.
3. Establezca una conexión con la base de datos en la que se van a crear las rutinas.

   ```
   db2 connect to <nombre-basedatos>
   ```

4. Precompile el archivo de código fuente con el mandato PREPARE.

   ```
   db2 prep myfile.sqC bindfile
   ```

   El precompilador generará salida que indicará si la precompilación se ha realizado satisfactoriamente o si se ha producido algún error. Este paso genera un bindfile denominado myfile.bnd que se puede utilizar para generar un paquete en el siguiente paso.
5. Enlace el paquete generado con la base de datos mediante el mandato BIND.

```
db2 bind myfile.bnd
```

El programa de utilidad bind generará salida que indicará si la vinculación se ha realizado satisfactoriamente o si se ha producido algún error.

6. Compile el archivo de código fuente mediante las opciones recomendadas de compilación y enlace:

   ```
   xlC_r -qstaticinline -I$HOME/sqllib/include -c $myfile.C
   ```

   El compilador generará salida si se produce algún error. Este paso genera un archivo de exportación denominado myfile.exp.

7. Enlace el archivo de código fuente para generar una biblioteca compartida.

   ```
   xlC_r -qmkshrobj -o $1 $1.o -L$ HOME/sqllib/include/lib32 -lDB2
   ```

   El enlazador generará salida si se produce algún error. Este paso genera un archivo de biblioteca compartida denominado myfile.

8. Copie la biblioteca compartida en el directorio de función de DB2 en el servidor de bases de datos.

   ```
   rm -f ~HOME/sqllib/function/myfile
   cp myfile $HOME/sqllib/function/myfile
   ```

   Este paso asegura que la biblioteca de la rutina está en el directorio por omisión en el que DB2 busca bibliotecas de rutinas. Consulte el tema sobre la creación de rutinas C y C++ para obtener más información sobre el despliegue de bibliotecas de rutinas.

9. Detenga y vuelva a iniciar la base de datos puesto que se trata de una recreación de un archivo de código fuente de rutina anteriormente creado.

   ```
   db2stop
   db2start
   ```

La creación de rutinas C y C++ suele resultar más sencilla si se utilizan los scripts de creación de ejemplo específicos del sistema operativo, que también se pueden utilizar como referencia para ver cómo crear rutinas desde la línea de mandatos.

Nota: La variable de entorno de sistema operativo `LD_LIBRARY_PATH` se ignora cuando se crean procedimientos de SQL incorporado.

Opciones de compilar y enlazar para rutinas C y C++

Opciones de compilación y enlace para rutinas C de AIX

Las siguientes son las opciones de compilación y enlace que recomienda DB2 para crear rutinas C (procedimientos almacenados y funciones definidas por el usuario) con el compilador C de AIX de IBM, como se muestra en el script de compilación `bldrtn`.

Opciones de compilación y enlace para bldrtn

Opciones de compilación:

xlc_r Utilizar la versión de varias hebras del compilador C de IBM, que es necesaria porque las rutinas se pueden ejecutar en el mismo proceso que otras rutinas (THREADSAFE) o en el propio motor (NOT FENCED).

$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_r Indica la utilización de la versión de varias hebras del compilador 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.

-qmkshrobj
Hace que se cree la biblioteca compartida.

-o $1 Especifica el nombre del archivo de salida.

$1.o Especifica el archivo objeto.

-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`.

-bE:$1.exp
Especifica un archivo de exportación. El archivo de exportación contiene una lista de las rutinas.

Consulte la documentación del compilador para conocer otras opciones de compilador.

Opciones de compilación y enlace para rutinas C++ de AIX

Las siguientes son las opciones de compilación y enlace que recomienda DB2 para crear rutinas C++ (procedimientos almacenados y funciones definidas por el usuario) con el compilador XL C/C++ de AIX deIBM como se muestra en el script de compilación `bldrtn`.

Opciones de compilación y enlace para bldrtn

Opciones de compilación:

xlC_r Utilizar la versión de varias hebras del compilador IBM XL C/C++, que es necesaria porque las rutinas se pueden ejecutar en el mismo proceso que otras rutinas (THREADSAFE) o en el propio motor (NOT FENCED).

$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_r Indica la utilización de la versión de varias hebras del compilador 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.

-qmkshrobj
Hace que se cree una biblioteca compartida.

-o $1 Especifica la salida como archivo de biblioteca compartida.

$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: `$HOME/sqllib/$LIB`. Si el usuario no especifica la opción `-L`, el compilador utiliza esta vía de acceso: `/usr/lib:/lib`.

-ldb2 Enlazar con la biblioteca de DB2.

-bE:$1.exp
Especifica un archivo de exportación. El archivo de exportación contiene una lista de las rutinas.

Consulte la documentación del compilador para conocer otras opciones de compilador.

Opciones de compilación y enlace para rutinas C de HP-UX

La tabla siguiente muestra las opciones de compilación y enlace que DB2 recomienda para crear rutinas en C (procedimientos almacenados y funciones definidas por el usuario) con el compilador C de HP-UX, tal como muestra el script de creación `bldrtn`.

Opciones de compilación y enlace para bldrtn

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**.

+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.

+u1 Permitir el acceso a datos no alineados. Sólo si la aplicación utiliza datos no alineados.

+z Generar código independiente de la posición.

-Ae Habilita la modalidad ampliada ANSI de HP.

-I$DB2PATH/include
Especifica la ubicación de los archivos de inclusión de DB2. Por ejemplo: `-I$DB2PATH/include`.

-D_POSIX_C_SOURCE=199506L
Opción de la biblioteca de hebras de POSIX que asegura que esté definido _REENTRANT, opción necesaria porque las rutinas se pueden ejecutar en el mismo proceso que otras rutinas (THREADSAFE) o en el propio motor (NOT FENCED).

-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:

ld Indica la utilización del enlazador para la edición de enlaces.

-b Crear una biblioteca compartida en lugar de un ejecutable normal.

-o $1 Especifica la salida como archivo de biblioteca compartida.

$1.o Especifica el archivo objeto del programa.

$EXTRA_LFLAG
Especificar la vía de acceso de tiempo de ejecución. Si se establece, para 32 bits contiene el valor "`+b$HOME/sqllib/lib32`", y para 64 bits: "`+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.

-lpthread
Establece un enlace con la biblioteca de hebras de POSIX.

Consulte la documentación del compilador para conocer otras opciones de compilador.

Opciones de compilación y enlace para rutinas C++ de HP-UX

La tabla siguiente muestra las opciones de compilación y enlace que DB2 recomienda para crear rutinas en C++ (procedimientos almacenados y funciones definidas por el usuario) con el compilador HP-UX C++, tal como muestra el script de creación `bldrtn`.

Opciones de compilación y enlace para bldrtn

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**.

+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.

+u1 Permite el acceso a datos no alineados.

+z Generar código independiente de la posición.

-ext Permite el uso de diversas extensiones de C++, tales como el soporte para "long long".

-mt Permite el soporte de hebras para el compilador aC++ de HP, que es necesario porque las rutinas se pueden ejecutaren el mismo proceso que otras rutinas (THREADSAFE) o en el propio motor (NOT FENCED).

-I$DB2PATH/include
Especifica la ubicación de los archivos de inclusión de DB2. Por ejemplo: `$DB2PATH/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**.

+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.

-mt Permite el soporte de hebras para el compilador aC++ de HP, que es necesario porque las rutinas se pueden ejecutaren el mismo proceso que otras rutinas (THREADSAFE) o en el propio motor (NOT FENCED).

-b Crear una biblioteca compartida en lugar de un ejecutable normal.

-o $1 Especifica el ejecutable.

$1.o Especifica el archivo objeto del programa.

$EXTRA_LFLAG
Especificar la vía de acceso de tiempo de ejecución. Si se establece, para 32 bits contiene el valor `-Wl,+b$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 rutinas C de Linux

La tabla siguiente muestra las opciones de compilación y enlace que DB2 recomienda para crear rutinas en C (procedimientos almacenados y funciones definidas por el usuario) con el compilador C de Linux, tal como muestra el script de creación `bldrtn`.

Opciones de compilación y enlace para bldrtn

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 32 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.

-D_REENTRANT
Define la opción _REENTRANT, que es necesaria pues las rutinas se pueden ejecutar en el mismo proceso que otras rutinas (THREADSAFE) o en el propio motor (NOT FENCED).

Opciones de enlace:

$CC El compilador gcc o xlc_r; hace que el compilador se utilice como interfaz del editor de enlaces.

$LINK_FLAGS
Contiene el valor "$EXTRA_C_FLAGS $SHARED_LIB_FLAG"

$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.

$SHARED_LIB_FLAG
Contiene `-shared` para el compilador gcc o `-qmkshrobj` para el compilador xlc_r.

-o $1 Especifica el ejecutable.

$1.o Incluir el archivo objeto del programa.

$EXTRA_LFLAG
Especifica la ubicación de las bibliotecas compartidas de DB2 durante la ejecución. Para 32 bits contiene el valor "`-Wl,-rpath,$DB2PATH/lib32`". 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.

-lpthread
Establece un enlace con la biblioteca de hebras de POSIX.

Consulte la documentación del compilador para conocer otras opciones de compilador.

Opciones de compilación y enlace para rutinas C++ de Linux

La tabla siguiente muestra las opciones de compilación y enlace que DB2 recomienda para crear rutinas en C++ (procedimientos almacenados y funciones definidas por el usuario) con el compilador C++ de Linux, tal como muestra el script de creación `bldrtn`.

Opciones de compilación y enlace para bldrtn

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 32 bits; o
- Ningún valor en Linux para IA64, para crear una biblioteca de 64 bits.

-fpic Genera código independiente de la posición.

-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.

-D_REENTRANT
Define la opción _REENTRANT, que es necesaria pues las rutinas se pueden ejecutar en el mismo proceso que otras rutinas (THREADSAFE) o en el propio motor (NOT FENCED).

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.

-shared
Genera una biblioteca compartida.

-o $1 Especifica el ejecutable.

$1.o Incluir el archivo objeto del programa.

$EXTRA_LFLAG
Especifica la ubicación de las bibliotecas compartidas de DB2 durante la ejecución. Para 32 bits contiene el valor "-Wl,-rpath,$DB2PATH/lib32". 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.

-lpthread
Establece un enlace con la biblioteca de hebras de POSIX.

Consulte la documentación del compilador para conocer otras opciones de compilador.

Opciones de compilación y enlace para rutinas C de Solaris

La tabla siguiente muestra las opciones de compilación y enlace que DB2 recomienda para crear rutinas en C (procedimientos almacenados y funciones definidas por el usuario) con el compilador C de Forte, tal como muestra el script de creación bldrtn.

Opciones de compilación y enlace para bldrtn

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 se establece del modo siguiente:

- "v8plusa": para aplicaciones de 32 bits en Solaris SPARC
- "v9": para aplicaciones de 64 bits en Solaris SPARC
- "sse2": para aplicaciones de 32 bits en Solaris x64
- "amd64": para aplicaciones de 64 bits en Solaris x64

-mt Permite el soporte de varias hebras, que es necesario pues las rutinas se pueden ejecutar en el mismo proceso que otras rutinas (THREADSAFE) o en el propio motor (NOT FENCED).

-DUSE_UI_THREADS
Permite el uso de las API de hebras "UNIX International" de Sun.

-Kpic Genera código independiente de la posición para bibliotecas compartidas.

-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 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 se establece del modo siguiente:

- "v8plusa": para aplicaciones de 32 bits en Solaris SPARC
- "v9": para aplicaciones de 64 bits en Solaris SPARC
- "sse2": para aplicaciones de 32 bits en Solaris x64
- "amd64": para aplicaciones de 64 bits en Solaris x64

-mt Esta opción es necesaria porque la biblioteca de DB2 se enlaza con `-mt`.

-G Genera una biblioteca compartida.

-o $1 Especifica el ejecutable.

$1.o Incluir el archivo objeto del programa.

-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 rutinas C++ de Solaris

La tabla siguiente muestra las opciones de compilación y enlace que DB2 recomienda para crear rutinas en C++ (procedimientos almacenados y funciones definidas por el usuario) con el compilador C++ de Forte, tal como muestra el script de creación bldrtn.

Opciones de compilación y enlace para bldrtn

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 se establece del modo siguiente:

- "v8plusa": para aplicaciones de 32 bits en Solaris SPARC
- "v9": para aplicaciones de 64 bits en Solaris SPARC
- "sse2": para aplicaciones de 32 bits en Solaris x64
- "amd64": para aplicaciones de 64 bits en Solaris x64

-mt Permite el soporte de varias hebras, que es necesario pues las rutinas se pueden ejecutar en el mismo proceso que otras rutinas (THREADSAFE) o en el propio motor (NOT FENCED).

-DUSE_UI_THREADS
Permite el uso de las API de hebras "UNIX International" de Sun.

-Kpic Genera código independiente de la posición para bibliotecas compartidas.

-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 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 se establece del modo siguiente:

- "v8plusa": para aplicaciones de 32 bits en Solaris SPARC
- "v9": para aplicaciones de 64 bits en Solaris SPARC
- "sse2": para aplicaciones de 32 bits en Solaris x64
- "amd64": para aplicaciones de 64 bits en Solaris x64

-mt Esta opción es necesaria porque la biblioteca de DB2 se enlaza con `-mt`.

-G Genera una biblioteca compartida.

-o $1 Especifica el ejecutable.

$1.o Incluir el archivo objeto del programa.

-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 rutinas C y C++ de Windows

A continuación se muestran las opciones de compilación y enlace recomendadas para DB2 para crear rutinas C y C++ (procedimientos almacenados y funciones definidas por el usuario) en Windows con el compilador Microsoft Visual C++, tal como se muestra en el archivo de proceso por lotes `bldrtn.bat`.

Opciones de compilación y enlace para bldrtn

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`, que indica el compilador Intel® C++ para aplicaciones de 32 ó 64 bits, o `ecl`, que indica el compilador Intel C++ para aplicaciones Itanium de 64 bits.
-Zi	Habilitar la información de depuración.
-Od	Inhabilitar la optimización.
-c	Hace que se efectúe solamente la compilación, no la edición de enlaces. La compilación y la edición de enlaces son pasos separados.
-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.
-MD	Enlace utilizando MSVCRT.LIB
Opciones de enlace:	
link	Indica la utilización del enlazador para la edición de enlaces.
-debug	Hace que se incluya información de depuración.
-out:%1.dll	Crear un archivo .DLL.
%1.obj	Incluir el archivo objeto.
db2api.lib	Enlazar con la biblioteca de DB2.
-def:%1.def	Archivo de definición de módulos.
Consulte la documentación del compilador para conocer otras opciones de compilador.	

Creación de procedimientos almacenados de SQL incorporado en C o C++ con archivos de configuración

El archivo de configuración, `stp.icc` en `sqllib/samples/c` y `sqllib/samples/cpp`, le permite crear procedimientos almacenados de SQL incorporado de DB2 en C y C++ en AIX.

Para utilizar el archivo de configuración a fin de crear la biblioteca compartida de procedimientos almacenados de SQL incorporado, `spserver`, a partir del archivo fuente `spserver.sqc`, siga estos pasos:

1. Asigne a la variable de entorno STP el nombre del programa, de este modo:
   - En el shell Bash o Korn:

```
export STP=spserver
```

- En el shell C:

```
setenv STP spserver
```

2. Si su directorio de trabajo contiene un archivo `stp.ics`, resultante de la creación de un programa diferente con el archivo `stp.icc`, suprima el archivo `stp.ics` mediante este mandato:

   ```
   rm stp.ics
   ```

   Si para el mismo programa que ahora creará de nuevo ya existe un archivo `stp.ics`, no es necesario suprimir el archivo.

3. Compile el programa de ejemplo, especificando:

   ```
   vacbld stp.icc
   ```

   Nota: El mandato `vacbld` está incluido en VisualAge C++.

La biblioteca compartida de procedimientos almacenados se copia en el directorio `sqllib/function` del servidor.

A continuación, catalogue los procedimientos almacenados de la biblioteca compartida ejecutando el script `spcat` en el servidor:

```
spcat
```

Este script conecta con la base de datos "sample", descataloga mediante `spdrop.db2` los procedimientos almacenados que se hubieran catalogado previamente, luego los cataloga llamando a `spcreate.db2`, y finalmente desconecta de la base de datos. Puede también ejecutar los scripts `spdrop.db2` y `spcreate.db2` por separado.

A continuación, detenga y rearranque la base de datos para que se reconozca la nueva biblioteca compartida. Si es necesario, establezca la modalidad de archivo para la biblioteca compartida a fin de que la instancia de DB2 pueda acceder a la biblioteca.

Una vez creada la biblioteca compartida de procedimientos almacenados, `spserver`, puede crear la aplicación cliente, `spclient`, la cual llama a los procedimientos almacenados contenidos en la biblioteca. Puede crear `spclient` utilizando el archivo de configuración, `emb.icc`.

Para invocar los procedimientos almacenados, ejecute la aplicación cliente de ejemplo, especificando lo siguiente: `spclient` *basedatos IDusuario contraseña*

donde

basedatos
: Es el nombre de la base de datos a la que desea conectarse. El nombre podría ser `sample`, su alias remoto u otro nombre.

IDusuario
: Es un ID de usuario válido.

contraseña
: Es una contraseña válida.

La aplicación cliente accede a la biblioteca compartida, `spserver`, la cual ejecuta diversas funciones de procedimiento almacenado contenidas en la base de datos del servidor. Los datos resultantes se devuelven a la aplicación cliente.

Creación de funciones definidas por el usuario en C o C++ con archivos de configuración (AIX)

El archivo de configuración udf.icc, contenido en sqllib/samples/c y sqllib/samples/cpp, le permite crear funciones definidas por el usuario en C o C++ en AIX.

Para utilizar el archivo de configuración a fin de crear el programa de función definida por el usuario udfsrv, a partir del archivo fuente udfsrv.c, siga estos pasos:

1. Asigne a la variable de entorno UDF el nombre del programa, de este modo:
   - En el shell Bash o Korn:
     ```
     export UDF=udfsrv
     ```
   - En el shell C:
     ```
     setenv UDF udfsrv
     ```
2. Si su directorio de trabajo contiene un archivo udf.ics, resultante de la creación de un programa diferente con el archivo udf.icc, suprima el archivo udf.ics mediante este mandato:
   ```
   rm udf.ics
   ```
   Si para el mismo programa que ahora creará de nuevo ya existe un archivo udf.ics, no es necesario suprimir el archivo.
3. Compile el programa de ejemplo, especificando:
   ```
   vacbld udf.icc
   ```

   Nota: El mandato vacbld está incluido en VisualAge C++.

La biblioteca de UDF se copia en el directorio sqllib/function del servidor.

Si es necesario, establezca la modalidad de archivo para la función definida por el usuario a fin de que la instancia de DB2 pueda ejecutar la función.

Una vez creado el programa udfsrv, puede crear la aplicación cliente, udfcli, la cual llama al programa. El programa se proporciona en versiones para CLI de DB2 y para SQL incorporado.

Puede crear el programa udfcli de la CLI de DB2, a partir del archivo fuente udfcli.c, contenido en sqllib/samples/cli, utilizando el archivo de configuración cli.icc.

Puede crear el programa udfcli de SQL incorporado, a partir del archivo fuente udfcli.sqc, contenido en sqllib/samples/c, utilizando el archivo de configuración emb.icc.

Para invocar la UDF, ejecute la aplicación solicitante de ejemplo entrando el nombre del ejecutable:

```
udfcli
```

La aplicación solicitante llama a la función ScalarUDF de la biblioteca udfsrv.

Reconstrucción de bibliotecas compartidas para rutinas DB2

DB2 situará en la antememoria, una vez cargadas, las bibliotecas compartidas utilizadas para procedimientos almacenados y funciones definidas por el usuario.

Si está desarrollando una rutina, puede que desee cargar varias veces la misma biblioteca, y esta utilización de la antememoria le impedirá obtener la última versión de una biblioteca compartida. La manera de evitar problemas derivados de la antememoria depende del tipo de rutina:

1. **Rutinas delimitadas, no protegidas por hebras.** La palabra clave KEEPFENCED de configuración del gestor de bases de datos tiene un valor por omisión de YES. Este valor mantiene vivo el proceso en modalidad delimitada. Este valor por omisión puede interferir con la recarga de la biblioteca. Es preferible cambiar el valor de esta palabra clave por NO mientras esté desarrollando rutinas delimitadas, sin protección por hebras, y luego revertir al valor YES cuando esté preparado para cargar la versión final de la biblioteca compartida. Para obtener más información, consulte "Actualización del archivo de configuración del gestor de bases de datos".
2. **Rutinas fiables o protegidas por hebras.** A excepción de las rutinas de SQL (incluidos los procedimientos de SQL), la única manera de asegurarse que se obtenga una versión actualizada de la biblioteca de rutinas de DB2 cuando se utilice dicha biblioteca para rutinas fiables o protegidas por hebras, consiste en reciclar la instancia de DB2 entrando db2stop seguido de db2start en la línea de mandatos. Esto no es necesario para una rutina de SQL puesto que, cuando se recrea, el compilador utiliza un nuevo nombre de biblioteca exclusivo para evitar posibles conflictos.

Para las rutinas que no son de SQL, también puede evitar los problemas derivados de la antememoria creando la nueva versión de la rutina con una biblioteca cuyo nombre sea distinto (por ejemplo, foo.a pasa a ser foo.1.a) y usando luego las sentencias ALTER PROCEDURE o ALTER FUNCTION de SQL con la nueva biblioteca.

Actualización del archivo de configuración del gestor de bases de datos

Este archivo contiene valores importantes para el desarrollo de aplicaciones.

La palabra clave `KEEPFENCED` tiene el valor por omisión `YES`. En el caso de las rutinas (procedimientos almacenados y UDF) sin protección de hebras, este valor mantiene vivo el proceso de la rutina. Es preferible cambiar el valor de esta palabra clave por `NO` mientras esté desarrollando estas rutinas, y luego revertir al valor `YES` cuando esté preparado para cargar la versión final de la biblioteca compartida. Para obtener más información, consulte "Reconstrucción de bibliotecas compartidas para rutinas DB2" en la página 284.

Nota: `KEEPFENCED` se conocía como `KEEPDARI` en versiones anteriores de DB2.

Para el desarrollo de aplicaciones Java, es necesario actualizar la palabra clave `JDK_PATH` con la vía de acceso donde está instalado el Java Development Kit.

Nota: `JDK_PATH` se conocía como `JDK11_PATH` en versiones anteriores de DB2.

Para cambiar estos valores, especifique:

```
db2 update dbm cfg using <palabra_clave> <valor>
```

Por ejemplo, para asignar el valor `NO` a la palabra clave `KEEPFENCED`:

```
db2 update dbm cfg using KEEPFENCED NO
```

Para establecer que el valor de la palabra clave JDK_PATH sea el directorio /home/db2inst/jdk13:

```
db2 update dbm cfg using JDK_PATH /home/db2inst/jdk13
```

Para ver los valores actuales contenidos en el archivo de configuración del gestor de bases de datos, especifique:

```
db2 get dbm cfg
```

Nota: En Windows, es necesario entrar estos mandatos en una ventana de mandatos de DB2.

Capítulo 7. Procedimientos COBOL

Los procedimientos COBOL se deben escribir de forma similar a los subprogramas COBOL.

Manejo de parámetros en un procedimiento COBOL

Cada parámetro que un procedimiento debe aceptar o pasar se debe declarar en LINKAGE SECTION. Por ejemplo, este fragmento de código corresponde a un procedimiento que acepta dos parámetros IN (un CHAR(15) y un INT) y pasa un parámetro OUT (un INT):

```
LINKAGE SECTION.
01  IN-SPERSON    PIC X(15).
01  IN-SQTY       PIC S9(9)  USAGE COMP-5.
01  OUT-SALESSUM  PIC S9(9)  USAGE COMP-5.
```

Asegúrese de que los tipos de datos COBOL que declare estén correctamente correlacionados con tipos de datos SQL. Para obtener una lista detallada de las correlaciones de tipos de datos entre SQL y COBOL, consulte "Tipos de datos SQL soportados en COBOL".

Cada parámetro se deberá listar entonces en PROCEDURE DIVISION. El ejemplo siguiente muestra una PROCEDURE DIVISION que corresponde a las definiciones de parámetros del ejemplo anterior de LINKAGE SECTION.

```
PROCEDURE DIVISION USING IN-SPERSON
                         IN-SQTY
                         OUT-SALESSUM.
```

Cómo salir de un procedimiento COBOL

Para salir correctamente del procedimiento, utilice los mandatos siguientes:

```
MOVE SQLZ-HOLD-PROC TO RETURN-CODE.
GOBACK.
```

Con estos mandatos, el procedimiento le devuelve correctamente a la aplicación cliente. Esto resulta especialmente importante cuando llama al procedimiento una aplicación cliente COBOL local.

Al crear un procedimiento COBOL, es muy recomendable que utilice el script de creación escrito para su sistema operativo y compilador. Los scripts de creación para Micro Focus COBOL se encuentran en el directorio sqllib/samples/cobol_mf. Los scripts de creación para IBM COBOL se encuentran en el directorio sqllib/samples/cobol.

A continuación, se muestra un ejemplo de procedimiento COBOL que acepta dos parámetros de entrada y luego devuelve un parámetro de salida y un conjunto de resultados:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.    "NEWSALE".
DATA DIVISION.

WORKING-STORAGE SECTION.
01  INSERT-STMT.
    05  FILLER   PIC X(24) VALUE "INSERT INTO SALES (SALES".
    05  FILLER   PIC X(24) VALUE "_PERSON,SALES) VALUES ('".
    05  SPERSON  PIC X(16).
    05  FILLER   PIC X(2) VALUE "',".
```

© Copyright IBM Corp. 1993, 2010

```
           05  SQTY     PIC S9(9).
           05  FILLER   PIC X(1) VALUE ")".
           EXEC SQL BEGIN DECLARE SECTION END-EXEC.
       01  INS-SMT-INF.
           05  INS-STMT.
           49  INS-LEN   PIC S9(4) USAGE COMP.
           49  INS-TEXT  PIC X(100).
       01  SALESSUM      PIC S9(9)  USAGE COMP-5.
           EXEC SQL END DECLARE SECTION END-EXEC.
           EXEC SQL INCLUDE SQLCA END-EXEC.

       LINKAGE SECTION.
       01  IN-SPERSON    PIC X(15).
       01  IN-SQTY       PIC S9(9)  USAGE COMP-5.
       01  OUT-SALESSUM  PIC S9(9)  USAGE COMP-5.

       PROCEDURE DIVISION USING IN-SPERSON
                                IN-SQTY
                                OUT-SALESSUM.
       MAINLINE.
           MOVE 0 TO SQLCODE.
           PERFORM INSERT-ROW.
           IF SQLCODE IS NOT EQUAL TO 0
              GOBACK
           END-IF.
           PERFORM SELECT-ROWS.
           PERFORM GET-SUM.
           GOBACK.
       INSERT-ROW.
           MOVE IN-SPERSON TO SPERSON.
           MOVE IN-SQTY TO SQTY.
           MOVE           INSERT-STMT TO INS-TEXT.
           MOVE LENGTH OF INSERT-STMT TO INS-LEN.
           EXEC SQL EXECUTE IMMEDIATE :INS-STMT END-EXEC.
       GET-SUM.
           EXEC SQL
              SELECT SUM(SALES) INTO :SALESSUM FROM SALES
           END-EXEC.
           MOVE SALESSUM TO OUT-SALESSUM.
       SELECT-ROWS.
           EXEC SQL
              DECLARE CUR CURSOR WITH RETURN FOR SELECT * FROM SALES
           END-EXEC.
           IF SQLCODE = 0
              EXEC SQL OPEN CUR END-EXEC
           END-IF.
```

La sentencia CREATE PROCEDURE que corresponde a este procedimiento es la siguiente:

```
CREATE PROCEDURE NEWSALE ( IN SALESPERSON CHAR(15),
                           IN SALESQTY INT,
                           OUT SALESSUM INT)
  RESULT SETS 1
  EXTERNAL NAME 'NEWSALE!NEWSALE'
  FENCED
  LANGUAGE COBOL
  PARAMETER STYLE SQL
  MODIFIES SQL DATA
```

La sentencia anterior presupone que la función COBOL se halla en una biblioteca denominada NEWSALE.

Nota: Al registrar un procedimiento COBOL en los sistemas operativos Windows, tome la precaución siguiente cuando identifique un cuerpo de procedimiento

almacenado en la cláusula EXTERNAL NAME de la sentencia CREATE. Si utiliza un ID de vía de acceso absoluto para identificar el cuerpo del procedimiento, debe añadir la extensión `.dll`. Por ejemplo:

```
CREATE PROCEDURE NEWSALE ( IN SALESPERSON CHAR(15),
                           IN SALESQTY INT,
                           OUT SALESSUM INT)
  RESULT SETS 1
  EXTERNAL NAME 'NEWSALE!NEWSALE'
  FENCED
  LANGUAGE COBOL
  PARAMETER STYLE SQL
  MODIFIES SQL DATA
  EXTERNAL NAME 'd:\mylib\NEWSALE.dll'
```

Soporte para el desarrollo de procedimientos externos en COBOL

Para desarrollar procedimientos externos en COBOL debe utilizar el software de desarrollo COBOL soportado.

Todo el software de desarrollo soportado para desarrollar aplicaciones de bases de datos en COBOL, se puede utilizar también para el desarrollo de procedimientos externos en COBOL.

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 38. 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
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

Tabla 38. 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
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[6] (456 ó 457)	01 name. 49 length PIC S9(4) COMP-5. 49 data PIC X(*n*). 32 673<=*n*<=32 700	Serie de caracteres de longitud variable larga
CLOB(*n*) (408 ó 409)	01 MY-CLOB USAGE IS SQL TYPE IS CLOB(n). 1<=*n*<=2 147 483 647	Serie de caracteres de longitud variable y objeto grande
Variable de localizador CLOB[4] (964 ó 965)	01 MY-CLOB-LOCATOR USAGE IS SQL TYPE IS CLOB-LOCATOR.	Identifica las entidades CLOB que residen en el servidor
Variable de referencia de archivo CLOB[4] (920 ó 921)	01 MY-CLOB-FILE USAGE IS SQL TYPE IS CLOB-FILE.	Descriptor de archivo que contiene datos CLOB
BLOB(*n*) (404 ó 405)	01 MY-BLOB USAGE IS SQL TYPE IS BLOB(n). 1<=*n*<=2 147 483 647	Serie binaria de longitud variable y objeto grande
Variable de localizador BLOB[4] (960 ó 961)	01 MY-BLOB-LOCATOR USAGE IS SQL TYPE IS BLOB-LOCATOR.	Identifica entidades BLOB que residen en el servidor
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

Tabla 38. 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
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 39. 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[6] (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 39. 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.
6. Los tipos de datos LONG VARCHAR y LONG VARGRAPHIC están en desuso y su utilización no es recomendable. Podrían eliminarse en un release futuro. En su lugar, utilice el tipo de datos CLOB o DBCLOB.

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.

Creación de rutinas COBOL

Opciones de compilar y enlazar para rutinas COBOL

Opciones de compilación y enlace para rutinas IBM COBOL de AIX

A continuación se muestran las opciones de compilación y enlace que se recomiendan para DB2 para crear rutinas COBOL (procedimientos almacenados) con el compilador IBM COBOL para AIX en AIX, tal como se muestra en el script de creación `bldrtn`.

Opciones de compilación y enlace para bldrtn

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.

-c Hace que se efectúe solamente la compilación, no la edición de enlaces. La compilación y la edición de enlaces son pasos separados.

-I$DB2PATH/include/cobol_a
Especifica la ubicación de los archivos de inclusión de DB2. Por ejemplo: `$HOME/sqllib/include/cobol_a`.

Opciones de enlace:

cob2 Indica la utilización del compilador para realizar la edición de enlaces.

-o $1 Especifica la salida como archivo de biblioteca compartida.

$1.o Especifica el archivo objeto del procedimiento almacenado.

checkerr.o
Hace que se incluya el archivo objeto de programa de utilidad para la comprobación de errores.

-bnoentry
Indica que no se especifique el punto de entrada por omisión de la biblioteca compartida.

-bE:$1.exp
Especifica un archivo de exportación. El archivo de exportación contiene una lista de los procedimientos almacenados.

-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 rutinas Micro Focus COBOL de AIX

A continuación se muestran las opciones de compilación y enlace que se recomiendan para DB2 para crear rutinas COBOL (procedimientos almacenados)

con el compilador Micro Focus COBOL en AIX, tal como se muestra en el script de creación `bldrtn`. 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 bldrtn

Opciones de compilación: **cob** El compilador MicroFocus COBOL. **-c** Hace que se efectúe solamente la compilación, no la edición de enlaces. La compilación y la edición de enlaces son pasos separados. **$EXTRA_COBOL_FLAG="-C MFSYNC"** Habilita el soporte de 64 bits. **-x** Compila y crea un módulo objeto cuando se utiliza con la opción `-c`.
Opciones de enlace: **cob** Hace que el compilador se utilice como interfaz del editor de enlaces. **-x** Hace que se cree una biblioteca compartida. **-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 rutinas Micro Focus COBOL de HP-UX

La tabla siguiente muestra las opciones de compilación y enlace que DB2 recomienda para crear rutinas COBOL (procedimientos almacenados) con el compilador Micro Focus COBOL en HP-UX, tal como muestra el script de creación `bldrtn`.

Opciones de compilación y enlace para bldrtn

Opciones de compilación: **cob** Indica el compilador COBOL. **$EXTRA_COBOL_FLAG** Contiene "-C MFSYNC" si la plataforma HP-UX es IA64 y el soporte de 64 bits está habilitado.

Opciones de enlace:

-y Especifica que la salida deseada es una biblioteca compartida.

-o $1 Especifica el ejecutable.

-L$DB2PATH/$LIB
Especifica la ubicación de las bibliotecas compartidas de tiempo de ejecución de DB2.

-ldb2 Enlazar con la biblioteca compartida 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 rutinas Micro Focus COBOL de Solaris

La tabla siguiente muestra las opciones de compilación y enlace que DB2 recomienda para crear rutinas COBOL (procedimientos almacenados) con el compilador Micro Focus COBOL en Solaris, tal como muestra el script de creación `bldrtn`.

Opciones de compilación y enlace para bldrtn

Opciones de compilación:

cob Indica el compilador 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.

-y Crea una biblioteca compartida autónoma auto-contenida.

-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/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 rutinas Micro Focus COBOL de Linux

A continuación se muestran las opciones de compilación y enlace que se recomiendan para DB2 para crear rutinas COBOL (procedimientos almacenados) con el compilador Micro Focus COBOL en Linux, tal como se muestra en el script de creación `bldrtn`.

Opciones de compilación y enlace para bldrtn

Opciones de compilación y enlace:

cob Indica el compilador COBOL

$EXTRA_COBOL_FLAG
Para el soporte de 64 bits, contiene el valor "-C MFSYNC"; de lo contrario, no contiene ningún valor.

-y Especifica compilar para objetos callable compartidos independientes

-o $1 Especifica el ejecutable.

$1.cbl Especifica el archivo fuente

-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.
Nota: Después de compilar los procedimientos almacenados, deben copiarse o enlazarse a `/usr/lib`. No basta con editar `ld.so.conf`.

Opciones de compilación y enlace para rutinas IBM COBOL de Windows

A continuación se muestran las opciones de compilación y enlace recomendadas para DB2 para crear rutinas COBOL (procedimientos almacenados y funciones definidas por el usuario) en Windows con el compilador IBM VisualAge COBOL, tal como se muestra en el archivo de proceso por lotes `bldrtn.bat`.

Opciones de compilación y enlace para bldrtn

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. El archivo de proceso por lotes tiene pasos separados para la compilación y la edición de enlaces.

-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 posee comentarios, el precompilador IBM 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.

Opciones de enlace:

ilink Utilizar el enlazador IBM VisualAge COBOL.

/free Formato libre.

/nol Sin logotipo.

/dll Crear la DLL con el nombre del programa fuente.

db2api.lib
Enlazar con la biblioteca de DB2.

%1.exp Incluir el archivo de exportación.

%1.obj Incluir el archivo objeto del programa.

iwzrwin3.obj
Incluir el archivo de objeto proporcionado por IBM VisualAge COBOL.

Consulte la documentación del compilador para conocer otras opciones de compilador.

Opciones de compilación y enlace para rutinas Micro Focus COBOL de Windows

La tabla siguiente muestra las opciones de compilación y enlace que DB2 recomienda para crear rutinas en COBOL (procedimientos almacenados y funciones definidas por el usuario) cuando se utiliza Windows y el compilador Micro Focus COBOL, tal como muestra el archivo de proceso por lotes `bldrtn.bat`.

Opciones de compilación y enlace para bldrtn

Opciones de compilación:	
cobol	Indica el compilador Micro Focus COBOL.
/case	Impide que los símbolos externos se conviertan en mayúsculas.
Opciones de enlace:	
cbllink	Utilizar el enlazador Micro Focus COBOL para la edición de enlaces.
/d	Crear un archivo .dll.
db2api.lib	Enlazar con la biblioteca DB2 de la API.
Consulte la documentación del compilador para conocer otras opciones de compilador.	

Creación de rutinas IBM COBOL en AIX

DB2 proporciona scripts de creación para compilar y enlazar programas COBOL de SQL incorporado y de la API 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 `bldrtn`, que reside en `sqllib/samples/cobol`, contiene los mandatos para crear rutinas (procedimientos almacenados). El script compila las rutinas y crea una biblioteca compartida que puede ser llamada por una aplicación cliente.

El primer parámetro, `$1`, especifica el nombre del archivo fuente. El segundo parámetro, `$2`, especifica el nombre de la base de datos a la que desea conectarse. Debido a que la biblioteca compartida se debe crear en la misma instancia donde reside la base de datos, no hay parámetros para el ID de usuario ni la contraseña.

Sólo es obligatorio el primer parámetro, el nombre del archivo fuente. El script utiliza el nombre del archivo fuente, `$1`, como nombre de la biblioteca compartida. El nombre de la base de datos es opcional. Si no se proporciona un nombre de base de datos, el programa utiliza la base de datos por omisión `sample`.

Si conecta con la base de datos "sample", especifique lo siguiente para crear el programa de ejemplo `outsrv` a partir del archivo fuente `outsrv.sqb`:

```
bldrtn outsrv
```

Si conecta con otra base de datos, especifique también el nombre de la base de datos:

```
bldrtn outsrv basedatos
```

El archivo de script copia la biblioteca compartida en el directorio `sqllib/function` del servidor.

Una vez creada la biblioteca compartida de la rutina, `outsrv`, puede crear la aplicación cliente, `outcli`, la cual llama a la rutina contenida en la biblioteca. Puede crear `outcli` utilizando el archivo de script `bldapp`.

Para invocar la rutina, ejecute la aplicación cliente de ejemplo, especificando lo siguiente:

```
outcli basedatos IDusuario contraseña
```

donde

basedatos
Es el nombre de la base de datos a la que desea conectarse. El nombre podría ser `sample`, su alias remoto u otro nombre.

IDusuario
Es un ID de usuario válido.

contraseña
Es una contraseña válida correspondiente al ID de usuario.

La aplicación cliente accede a la biblioteca compartida, `outsrv`, la cual ejecuta la rutina del mismo nombre contenida en la base de datos del servidor, y devuelve los datos resultantes a la aplicación cliente.

Creación de rutinas Micro Focus COBOL de UNIX

Los productos de base de datos DB2 proporcionan scripts de compilación para compilar y enlazar programas de SQL incorporado de COBOL de Micro Focus 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.

Acerca de esta tarea

El script bldrtn contiene los mandatos para crear rutinas (procedimientos almacenados). El script compila el archivo fuente de la rutina y crea una biblioteca compartida que puede ser llamada por una aplicación cliente.

El primer parámetro, `$1`, especifica el nombre del archivo fuente. El script utiliza el nombre del archivo fuente como nombre de la biblioteca compartida. El segundo parámetro, `$2`, especifica el nombre de la base de datos a la que desea conectarse. Debido a que la biblioteca compartida se debe crear en la misma instancia donde reside la base de datos, no hay parámetros para el ID de usuario ni la contraseña.

Sólo es obligatorio el primer parámetro, el nombre del archivo fuente. El nombre de la base de datos es opcional. Si no se proporciona un nombre de base de datos, el programa utiliza la base de datos por omisión `sample`.

Procedimiento

1. Antes de crear las rutinas de Micro Focus COBOL, debe ejecutar estos mandatos:

   ```
   db2stop
   db2set DB2LIBPATH=$LD_LIBRARY_PATH
   db2set DB2ENVLIST="COBDIR LD_LIBRARY_PATH"
   db2set
   db2start
   ```

   Asegúrese de que db2stop detiene la base de datos. El último mandato db2set se emite para comprobar los valores: compruebe que **DB2LIBPATH** y **DB2ENVLIST** estén definidos correctamente.
2. Si conecta con la base de datos de ejemplo, especifique lo siguiente para crear el programa de ejemplo outsrv a partir del archivo fuente `outsrv.sqb`: `bldrtn outsrv`.

   Si conecta con otra base de datos, especifique también el nombre de la base de datos. Por ejemplo:

   ```
   bldrtn outsrv basedatos
   ```

El archivo de script copia la biblioteca compartida en el directorio `sqllib/function` del servidor.

Una vez creado el procedimiento almacenado outsrv, puede crear la aplicación cliente outcli que llama al procedimiento almacenado. Puede crear outcli utilizando el archivo de script bldapp.

3. Para invocar el procedimiento almacenado, ejecute la aplicación cliente de ejemplo, especificando lo siguiente: `outcli` *`basedatos IDusuario contraseña`*

   donde

   basedatos
   Es el nombre de la base de datos a la que desea conectarse. El nombre podría ser `sample`, o su alias, u otro nombre.

   IDusuario
   Es un ID de usuario válido.

   contraseña
   Es una contraseña válida correspondiente al ID de usuario.

   La aplicación cliente accede a la biblioteca compartida, `outsrv`, la cual ejecuta la función de procedimiento almacenado del mismo nombre contenida la base de datos el servidor. A continuación los datos resultantes se devuelven a la aplicación cliente.

Creación de rutinas IBM COBOL en Windows

DB2 proporciona archivos de creación para compilar y enlazar programas de la API de DB2 y de SQL incorporado en IBM COBOL. 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 `bldrtn.bat` contiene los mandatos para crear rutinas (procedimientos almacenados) de SQL incorporado. El archivo de proceso por lotes compila las rutinas y crea una DLL en el servidor. Utiliza dos parámetros como entrada, que están representados dentro del archivo de proceso por lotes por las variables `%1` y `%2`.

El primer parámetro, `%1`, especifica el nombre del archivo fuente. El archivo de proceso por lotes utiliza el nombre del archivo fuente, `%1`, como nombre de la DLL. El segundo parámetro, `%2`, especifica el nombre de la base de datos a la que desea conectarse. Debido a que el procedimiento almacenado se debe crear en la misma instancia donde reside la base de datos, no hay parámetros para el ID de usuario ni la contraseña.

Sólo es obligatorio el primer parámetro, el nombre del archivo fuente. El nombre de la base de datos es opcional. Si no se proporciona un nombre de base de datos, el programa utiliza la base de datos por omisión `sample`.

Si utiliza el precompilador de DB2 por omisión, `bldrtn.bat` pasa los parámetros al archivo de precompilación y vinculación, `embprep.bat`.

Si se 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.

Si conecta con la base de datos "sample", especifique lo siguiente para crear el programa de ejemplo `outsrv` a partir del archivo fuente `outsrv.sqb`:

```
bldrtn outsrv
```

Si conecta con otra base de datos, especifique también el nombre de la base de datos:

```
bldrtn outsrv basedatos
```

El archivo de proceso por lotes copia la DLL en la vía de acceso `sqllib\function` del servidor.

Una vez creada la DLL `outsrv`, puede crear la aplicación cliente `outcli`, la cual llama a la rutina contenida en la DLL y que tiene el mismo nombre que la DLL. Puede crear `outcli` con el archivo por lotes `bldapp.bat`.

Para invocar la rutina `outsrv`, ejecute la aplicación cliente de ejemplo, especificando lo siguiente:

```
outcli basedatos IDusuario contraseña
```

donde

basedatos
Es el nombre de la base de datos a la que desea conectarse. El nombre podría ser `sample`, su alias remoto u otro nombre.

IDusuario
Es un ID de usuario válido.

contraseña
Es una contraseña válida correspondiente al ID de usuario.

La aplicación cliente accede a la DLL `outsrv`, la cual ejecuta la rutina del mismo nombre contenida en la base de datos del servidor, y devuelve los datos resultantes a la aplicación cliente.

Creación de rutinas Micro Focus COBOL en Windows

DB2 proporciona archivos de creación para compilar y enlazar programas de la API de DB2 y de SQL incorporado en Micro Focus COBOL. 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 `bldrtn.bat` contiene los mandatos para crear rutinas (procedimientos almacenados) de SQL incorporado. El archivo de proceso por lotes compila las rutinas y crea una DLL en el servidor. El archivo utiliza dos parámetros como entrada, que están representados dentro del archivo de proceso por lotes por las variables `%1` y `%2`.

El primer parámetro, `%1`, especifica el nombre del archivo fuente. El archivo de proceso por lotes utiliza el nombre del archivo fuente, `%1`, como nombre de la DLL. El segundo parámetro, `%2`, especifica el nombre de la base de datos a la que desea

conectarse. Debido a que el procedimiento almacenado se debe crear en la misma instancia donde reside la base de datos, no hay parámetros para el ID de usuario ni la contraseña.

Sólo es obligatorio el primer parámetro, el nombre del archivo fuente. El nombre de la base de datos es opcional. Si no se proporciona un nombre de base de datos, el programa utiliza la base de datos por omisión `sample`.

Si conecta con la base de datos "sample", especifique lo siguiente para crear el programa de ejemplo `outsrv` a partir del archivo fuente `outsrv.sqb`:

```
bldrtn outsrv
```

Si conecta con otra base de datos, especifique también el nombre de la base de datos:

```
bldrtn outsrv basedatos
```

El archivo de script copia la DLL en el directorio `sqllib/function` del servidor.

Una vez creada la DLL `outsrv`, puede crear la aplicación cliente `outcli`, la cual llama a la rutina contenida en la DLL y que tiene el mismo que la DLL. Puede crear `outcli` utilizando el archivo de proceso por lotes `bldapp.bat`.

Para invocar la rutina `outsrv`, ejecute la aplicación cliente de ejemplo, especificando lo siguiente:

```
outcli basedatos IDusuario contraseña
```

donde

basedatos
Es el nombre de la base de datos a la que desea conectarse. El nombre podría ser `sample`, o su alias, u otro nombre.

IDusuario
Es un ID de usuario válido.

contraseña
Es una contraseña válida correspondiente al ID de usuario.

La aplicación cliente accede a la DLL `outsrv`, la cual ejecuta la rutina del mismo nombre contenida en la base de datos del servidor. A continuación los datos resultantes se devuelven a la aplicación cliente.

Capítulo 8. Rutinas Java

Las rutinas Java son rutinas externas que tienen una implementación de lenguaje de programación Java. Las rutinas Java se crean en una base de datos ejecutando una sentencia CREATE PROCEDURE o CREATE FUNCTION. Esta sentencia debe indicar que la rutina se implementa en Java con la cláusula LANGUAGE JAVA. También debe especificarse con la cláusula EXTERNAL, la clase Java que la implementa.

Los procedimientos, funciones y métodos externos se pueden crear en Java.

Las rutinas Java pueden ejecutar sentencias de SQL.

Los términos siguientes son importantes en el contexto de las rutinas Java:

JDBC Interfaz de programación de aplicaciones para acceder a datos de bases de datos que permite la ejecución de SQL dinámico.

SQLJ Interfaz de programación de aplicaciones (API) para acceder a datos de bases de datos que permite la ejecución de SQL dinámico y estático.

Sentencia CREATE
La sentencia CREATE del lenguaje SQL utilizada para crear la rutina en la base de datos.

Código fuente del cuerpo de la rutina
Archivo de código fuente que contiene la implementación de la rutina Java. La rutina Java puede acceder a la base de datos utilizando interfaces de programación de aplicaciones JDBC o SQLJ.

JDBC Interfaz de programación de aplicaciones que permite la ejecución de sentencias de SQL dinámico en código Java.

SQLJ Interfaz de programación de aplicaciones que permite la ejecución de sentencias de SQL estático en código Java.

SDK para Java
Kit de desarrollo de software para Java que se proporciona y es necesario para la compilación del código fuente Java.

Clase de rutina
Archivo de código fuente Java que contiene la forma compilada del código fuente de la rutina Java. Los archivos de clases Java pueden ser independientes o pueden formar parte de una colección de archivos de clases Java dentro de un archivo JAR.

Antes de desarrollar una rutina Java, es importante comprender los conceptos básicos de las rutinas y las características exclusivas de las rutinas Java. También es importante tener conocimientos del desarrollo de aplicaciones de bases de datos en Java utilizando JDBC o bien SQLJ. Para obtener más información sobre estas cuestiones, consulte los temas:

- "Creación de rutinas externas" en la página 120
- "Desarrollo de aplicaciones Java para DB2" en *Desarrollo de aplicaciones Java*

© Copyright IBM Corp. 1993, 2010

Software de desarrollo de rutinas Java soportado

Para desarrollar y desplegar rutinas externas en Java, debe utilizar el software de desarrollo Java soportado.

La versión del kit de desarrollo de software (SDK) mínima para el desarrollo de la rutina Java es la siguiente:

- IBM SDK para Java 1.4.2

La versión de SDK máxima soportada de SDK para el desarrollo de rutinas Java es:

- IBM SDK para Java 6

  IBM SDK para Java 6 admite procedimientos almacenados definidos por el usuario en estos sistemas operativos:

  – AIX 5
  – HP-UX para sistemas basados en Itanium
  – Linux en x86
  – Linux en AMD64/EM64T
  – Linux en POWER
  – Windows en x86
  – Windows en x64, para AMD64/EM64T

Se recomienda utilizar el SDK para Java instalado con el producto DB2 Database para Linux, UNIX, y Windows; y, sin bien, se puede especificar un SDK alternativo para Java, el SDK debe tener la misma anchura de bit que la instancia de DB2.

Todo software de desarrollo que esté soportado para el desarrollo de aplicaciones de bases de datos DB2 en Java se puede usar para el desarrollo de rutinas externas en Java.

Soporte de interfaces de programación de aplicaciones JDBC y SQLJ para rutinas Java

Las rutinas externas desarrolladas en Java pueden hacer uso de las siguientes interfaces de programación de aplicaciones (API):

- JDBC
- SQLJ

Empezando por DB2 versión 9.5, el controlador predeterminado es IBM Data Server Driver para JDBC y SQLJ. Da soporte tanto a las API de JCBC como SQLJ y se puede utilizar para desarrollar rutinas externas de Java.

El controlador IBM Data Server Type 2 también se puede utilizar para el desarrollo de rutinas Java, pero sólo proporciona soporte para JDBC Type 2. Se recomienda muy especialmente utilizar IBM Data Server Driver para JDBC y SQLJ ya que IBM Data Server Type 2 está obsoleto.

Los procedimientos para implementar las rutinas de Java son las mismas, independientemente de qué API o controlador se utilice.

Especificación de un SDK para el desarrollo de rutinas Java (UNIX)

Para crear y ejecutar un código de rutina Java en entornos UNIX, el parámetro de configuración del gestor de bases de datos DB2, el parámetro de configuración del gestor JDK_PATH debe establecerse en la vía de acceso de un SDK para Java que está instalado en el servidor de bases de datos DB2. Si el proceso de instalación del producto DB2 Database para Linux, UNIX, y Windows instala un SDK para Java, por omisión, el parámetro JDK_PATH se establece en la vía de acceso de dicho SDK para Java:

- Servidores de bases de datos `$INSTDIR/sqllib/java/jdk32` para DB2 instalados en Linux en entornos x86
- `$INSTDIR/sqllib/java/jdk64` para servidores de base de datos DB2 que proporcionen un SDK instalado en todos los demás entornos

El valor del parámetro JDK_PATH se puede cambiar para especificar otro SDK para Java instalado en el sistema, aunque el SDK debe tener la misma anchura de bit que la instancia de DB2.

Requisitos previos

- El usuario debe tener acceso al servidor de bases de datos DB2.
- El usuario debe tener autorización para leer y actualizar el archivo de configuración del gestor de base de datos.
- El usuario debe tener autorización para instalar un SDK para Java en el sistema de archivos donde está instalada la instancia de DB2.

Procedimiento

1. Compruebe el valor del parámetro JDK_PATH emitiendo el mandato siguiente desde una ventana de mandatos de DB2.

   ```
   db2 get dbm cfg
   ```

   Es posible que desee redirigir la salida a un archivo, para tener una visión más fácil. El valor del parámetro JDK_PATH aparece cerca del comienzo de los datos de salida.
2. Si desea utilizar un SDK diferente para Java, instálelo en el servidor de bases de datos DB2 y anote la vía de acceso de instalación. Necesitará el valor de dicha vía de acceso de instalación para el siguiente paso.
3. Actualice el valor del parámetro JDK_PATH emitiendo el siguiente mandato desde una ventana de mandatos de DB2 donde `<path>` es la vía de acceso donde está instalado el otro SDK de Java:

   ```
   db2 update dbm cfg using JDK_PATH <vía_acceso>
   ```
4. Detenga y reinicie la instancia de DB2 emitiendo los siguientes mandatos de DB2 desde una ventana de mandatos de DB2:

   ```
   db2stop;

   db2start;
   ```
5. Compruebe si el valor del parámetro JDK_PATH se ha establecido correctamente emitiendo el siguiente mandato desde una ventana de mandatos de DB2:

   ```
   db2 get dbm cfg
   ```

Cuando haya completado estos pasos, el SDK especificado para Java se utilizará para crear y ejecutar rutinas Java. Las variables de entorno CLASSPATH, PATH y LIBPATH del entorno se establecen de modo automático.

Especificación de un controlador para rutinas Java

El desarrollo y la invocación de rutinas Java requiere que se especifique un controlador JDBC o SQLJ. Las rutinas Java pueden utilizar uno de los dos controladores siguientes:

- IBM Data Server Driver para JDBC y SQLJ Versión 3.0
- DB2 Type 2 Driver

En IBM Data Server Driver para JDBC y SQLJ Versión 4.0, `db2jcc4.jar` incluye varias características de JDBC Versión 4.0. El controlador está soportado en DB2 Versión 9.5 y posterior. Sin embargo, las rutinas Java no soportan IBM Data Server Driver para JDBC y SQLJ Versión 4.0.

Por omisión, DB2 utiliza IBM Data Server Driver para JDBC y SQLJ. Este controlador es preferible porque es más robustos, DB2 Type 2 Driver está obsoleto y porque es un requisito previo si las rutinas Java contienen:

- Parámetros de tipo de datos XML
- Variables de tipo de datos XML
- Referencias a datos XML
- Referencias a funciones XML
- Cualquier otra característica de XML nativo

Si se producen problemas en la migración de rutinas Java existentes, la variable de entorno `DB2_USE_DB2JCCT2_JROUTINE` DB2 se puede establecer en el valor, `NO` para utilizar el controlador IBM DB2 Type 2 Driver ya existente. Esta operación puede realizarse emitiendo el mandato siguiente desde una ventana de mandatos de DB2:

```
db2set DB2_USE_DB2JCCT2_JROUTINE=NO
```

Después de emitir el mandato, debe detener y reiniciar la instancia de DB2 para que el cambio entre en vigor.

Herramientas para el desarrollo de rutinas Java (JDBC y SQLJ)

Las herramientas hacen que el desarrollo de rutinas de Java sea rápido y fácil. La siguiente herramienta de DB2 proporciona soporte de interfaz gráfica de usuario para desarrollar, depurar y desplegar rutinas de Java:

- IBM Data Studio

Las siguientes interfaces de mandatos también se pueden utilizar para desarrollar, depurar y desplegar rutinas de Java:

- Procesador de línea de mandatos de DB2
- Ventana de mandatos de DB2

Otros productos de software de IBM ofrecen herramientas gráficas para desarrollar rutinas de Java, entre otras:

- IBM Rational Application Developer
- Distributed Unified Debugger

Diseño de rutinas Java

El diseño de rutinas Java es una tarea que debe preceder a la creación de dichas rutinas Java. El diseño de rutinas Java está relacionado tanto con el diseño de rutinas externas implementadas en otros lenguajes de programación como en el diseño de aplicaciones de base de datos Java.

Conocimientos y experiencia en el desarrollo de aplicaciones de SQL incorporado, así como un conocimiento general de las rutinas externas. Los temas siguientes proporcionan parte de la información necesaria sobre requisitos previos.

Para obtener más información sobre las funciones y usos de las rutinas externas, consulte:

- "Implementación de rutinas externas" en la página 22

Para obtener más información sobre el modo de programar una aplicación Java básica utilizando las interfaces de programación de aplicaciones JDBC o SQLJ, consulte los temas siguientes:

- "Ejemplo de una aplicación JDBC simple" en *Desarrollo de aplicaciones Java*
- "Ejemplo de una aplicación SQLJ simple" en *Desarrollo de aplicaciones Java*

Con el conocimiento de los requisitos previos, el diseño de rutinas Java consiste principalmente en conocer las funciones y características exclusivas de las rutinas Java:

- "Tipos de datos de SQL soportados en rutinas Java"
- "Parámetros de las rutinas Java" en la página 310
- "Estilo de parámetro en procedimientos JAVA" en la página 310
- "Funciones y métodos Java con parámetros de estilo JAVA" en la página 312
- "Devolución de conjuntos de resultados desde procedimientos JDBC" en la página 322
- "Devolución de conjuntos de resultados desde procedimientos de SQLJ" en la página 323
- "Restricciones de las rutinas Java" en la página 326
- "Modelo de ejecución de las funciones para Java" en la página 144

Después de conocer las características de Java, puede interesarle consultar lo siguiente:

- "Creación de rutinas Java desde la línea de mandatos" en la página 328

Tipos de datos de SQL soportados en rutinas Java

Los tipos de datos del lenguaje de programación Java deben utilizarse en código fuente Java para almacenar valores de tipo de datos SQL según la especificación de la interfaz de programación de aplicaciones JDBC y SQLJ. El controlador DB2 para JDBC y SQLJ convierte los datos intercambiados entre el código fuente Java y una base de datos DB2 según correlaciones de tipos de datos específicos.

Las correlaciones de datos son válidas para:

- Aplicaciones de bases de datos Java
- Rutinas Java definidas e implementadas mediante PARAMETER STYLE JAVA

Los tipos de datos Java que se correlacionan con tipos de datos SQL son los siguientes:

Tabla 40. Tipos de datos de SQL correlacionados con declaraciones Java

Tipo de columna SQL	Tipo de datos de Java	Descripción del tipo de columna de SQL
SMALLINT (500 ó 501)	short, boolean	Entero con signo de 16 bits
INTEGER (496 ó 497)	int	Entero con signo de 32 bits
BIGINT [1] (492 ó 493)	long	Entero con signo de 64 bits
REAL (480 ó 481)	float	Coma flotante de precisión simple
DOUBLE (480 ó 481)	double	Coma flotante de precisión doble
DECIMAL(*p*, *s*) (484 ó 485)	java.math.BigDecimal	Decimal empaquetado
CHAR(*n*) (452 ó 453)	java.lang.String	Serie de caracteres de longitud fija con longitud *n* donde *n* va de 1 a 254
CHAR(*n*) FOR BIT DATA	byte[]	Serie de caracteres de longitud fija con longitud *n* donde *n* va de 1 a 254
VARCHAR(*n*) (448 ó 449)	java.lang.String	Serie de caracteres de longitud variable
VARCHAR(*n*) FOR BIT DATA	byte[]	Serie de caracteres de longitud variable
LONG VARCHAR (456 ó 457)	java.lang.String	Serie de caracteres de longitud variable larga
LONG VARCHAR FOR BIT DATA	byte[]	Serie de caracteres de longitud variable larga
BLOB(*n*) (404 ó 405)	java.sql.Blob	Serie binaria de longitud variable y objeto grande
CLOB(*n*) (408 ó 409)	java.sql.Clob	Serie de caracteres de longitud variable y objeto grande
DBCLOB(*n*) (412 ó 413)	java.sql.Clob	Serie de caracteres de doble byte de longitud variable y objeto grande

Tabla 40. Tipos de datos de SQL correlacionados con declaraciones Java (continuación)

Tipo de columna SQL	Tipo de datos de Java	Descripción del tipo de columna de SQL
DATE (384 ó 385)	java.sql.Date	Serie de caracteres de 10 bytes
TIME (388 ó 389)	java.sql.Time	Serie de caracteres de 8 bytes
TIMESTAMP (392 ó 393)	java.sql.Timestamp	Serie de caracteres de 26 bytes
GRAPHIC(*n*) (468 ó 469)	java.lang.String	Serie de caracteres de doble byte y longitud fija
VARGRAPHIC(*n*) (464 ó 465)	java.lang.String	Serie de caracteres de doble byte variable no terminada en nulo con indicador de longitud de serie de 2 bytes
LONGVARGRAPHIC (472 ó 473)	java.lang.String	Serie de caracteres de doble byte variable no terminada en nulo con indicador de longitud de serie de 2 bytes
XML(*n*) (408 ó 409)	java.sql.Clob	El tipo de datos XML se representa del mismo modo que un tipo de datos CLOB; es decir, una serie de caracteres de longitud variable de objetos grandes
ARRAY	java.sql.Array	Matriz de datos de SQL.

Nota:

1. Respecto a las aplicaciones Java conectadas desde un cliente DB2 Universal Database versión 8.1 a un servidor DB2 Universal Database versión 7.1 (o 7.2), tenga en cuenta lo siguiente: cuando se utiliza el método getObject() para recuperar un valor BIGINT, se devuelve un objeto java.math.BigDecimal.
2. Los parámetros de un tipo de datos de matriz de SQL se correlacionan con la clase com.ibm.db2.ARRAY.
3. LONG VARCHAR, LONG VARGRAPHIC, XML, REFERENCE, UDT y ARRAY no están soportados para el tipo de datos ARRAY.
4. Se sigue ofreciendo soporte para los tipos de datos LONG VARCHAR y LONG VARGRAPHIC, pero éstos se han dejado de utilizar y no se recomiendan ya que puede que se eliminen en un release futuro.

Contextos de conexión en rutinas SQLJ

Con la introducción de rutinas de varias hebras en DB2 Universal Database, Versión 8, es importante que las rutinas de SQLJ eviten la utilización del contexto de conexión por omisión. Es decir, cada sentencia de SQL debe indicar explícitamente el objeto de ConnectionContext (contexto de conexión) y dicho contexto debe replicarse de modo explícito en el método Java. Por ejemplo, en los anteriores releases de DB2, una rutina de SQLJ podía escribirse del siguiente modo:

```
class myClass
{
  public static void myRoutine( short myInput )
  {
```

```
    DefaultContext ctx = DefaultContext.getDefaultContext();
    #sql { some SQL statement };
  }
}
```

Esta utilización del contexto por omisión hace que todas las hebras de un entorno con varias hebras utilicen el mismo contexto de conexión, el cual, a su vez, dará como resultado que se produzcan anomalías inesperadas.

La rutina SQLJ anterior debe cambiarse del siguiente modo:

```
#context MyContext;

class myClass
{
  public static void myRoutine( short myInput )
  {
    MyContext ctx = new MyContext( "jdbc:default:connection", false );
    #sql [ctx] { some SQL statement };
    ctx.close();
  }
}
```

De este modo, cada invocación de la rutina creará su propio ConnectionContext (Contexto de conexión) exclusivo (y su propia conexión JDBC subyacente), lo cual evitará interferencias inesperadas por parte de hebras simultáneas.

Parámetros de las rutinas Java

La declaración de parámetros de las rutinas Java debe cumplir con los requisitos de uno de los estilos de parámetros soportados. Se da soporte a los dos siguientes estilos de parámetros para las rutinas Java:

- PARAMETER STYLE JAVA
- PARAMETER STYLE DB2GENERAL

Le recomendamos encarecidamente que especifique la cláusula PARAMETER STYLE JAVA en la sentencia CREATE de la rutina. Con PARAMETER STYLE JAVA, una rutina utilizará un convenio de pase de parámetros que esté en conformidad con la especificación del lenguaje Java y de las rutinas SQLJ.

Existen algunas características de las rutinas Java que no se pueden implementar ni utilizar con PARAMETER STYLE JAVA. Son las siguientes:

- funciones de tabla
- áreas reutilizables en funciones
- acceso a la estructura DBINFO en funciones
- la posibilidad de efectuar una FINAL CALL (y una primera llamada separada) a una función o método

Si tiene necesidad de implementar las características anteriores, puede implementar su rutina en C o escribirla en Java, utilizando el estilo de parámetro DB2GENERAL.

Aparte de estos casos específicos, siempre debe crear e implementar las rutinas Java con PARAMETER STYLE JAVA.

Estilo de parámetro en procedimientos JAVA

El estilo de parámetro recomendado para implementaciones de procedimientos Java es PARAMETER STYLE JAVA.

La signatura de los procedimientos almacenados con PARAMETER STYLE JAVA sigue este formato:

```
public static void nombre-método ( argumentos-SQL,
ResultSet[] matriz-conjuntos-resultados )
                                throws SQLException
```

nombre-método
: Nombre del método. Durante el registro de la rutina, se especifica este valor junto con el nombre de clase en la cláusula EXTERNAL NAME de la sentencia CREATE PROCEDURE.

argumentos-SQL
: Corresponde a la lista de parámetros de entrada de la sentencia CREATE PROCEDURE. Los parámetros de modalidad OUT o INOUT se pasan como matrices de un solo elemento. Para cada conjunto de resultados que se especifique en la cláusula DYNAMIC RESULT SETS de la sentencia CREATE PROCEDURE, se añade a la lista de parámetros una matriz de un solo elemento del tipo ResultSet.

matriz-conjunto-resultados
: Nombre de la matriz de objetos ResultSet. Por cada conjunto de resultados declarado en el parámetro DYNAMIC RESULT SETS de la sentencia CREATE PROCEDURE, se debe declarar un parámetro de tipo ResultSet[] en la signatura del método Java.

El siguiente es un ejemplo de un procedimiento almacenado Java que acepta un parámetro de entrada y devuelve un parámetro de salida y un conjunto de resultados:

```
public static void javastp( int inparm, int[] outparm, ResultSet[] rs )
                  throws SQLException
{
  Connection con = DriverManager.getConnection( "jdbc:default:connection");
  PreparedStatement stmt = null;
  String sql = "SELECT value FROM table01 WHERE index = ?";

  //Preparar la consulta con el valor del índice
  stmt = con.prepareStatement( sql );
  stmt.setInt( 1, inparm );

  //Ejecutar la consulta y establecer el parámetro de salida
  rs[0] = stmt.executeQuery();
  outparm[0] = inparm + 1;

  //Cerrar los recursos abiertos
  if (stmt != null) stmt.close();
  if (con != null) con.close();

  return;
}
```

La sentencia CREATE PROCEDURE correspondiente a este procedimiento almacenado es la siguiente:

```
CREATE PROCEDURE javaproc( IN in1 INT, OUT out1 INT )
  LANGUAGE java
  PARAMETER STYLE java
  DYNAMIC RESULT SETS 1
  FENCED THREADSAFE
  EXTERNAL NAME 'myjar:stpclass.javastp'
```

La sentencia anterior supone que el método se halla en una clase denominada `stpclass`, ubicada en un archivo JAR catalogado en la base de datos con el ID de Jar `myjar`.

Nota:

1. Las rutinas con PARAMETER STYLE JAVA utilizan excepciones para pasar los datos sobre errores al invocador. Si desea información completa, lo que incluye la pila de llamadas de excepción, consulte el `registro de notificaciones de administración`. Aparte de este detalle, no existe ninguna otra consideración especial para invocar las rutinas con PARAMETER STYLE JAVA.
2. Las llamadas JNI no están soportadas en las rutinas Java. Sin embargo, es posible invocar funciones de C desde rutinas Java, anidando una invocación de una rutina C. Esto implica mover la función de C deseada a una rutina, registrarla e invocarla desde dentro de la rutina Java.

Funciones y métodos Java con parámetros de estilo JAVA

El estilo de parámetro recomendado para funciones y métodos Java es PARAMETER STYLE JAVA.

La signatura de las funciones y métodos PARAMETER STYLE JAVA tiene este formato:

```
public static tipo-retorno nombre-método ( argumentos-SQL ) throws SQLException
```

tipo-retorno
: Tipo de datos del valor que la rutina escalar debe devolver. En la rutina, el valor de retorno se devuelve al invocador a través de una sentencia de retorno.

nombre-método
: Nombre del método. Durante el registro de la rutina, se especifica este valor junto con el nombre de clase en la cláusula EXTERNAL NAME de la sentencia CREATE de la rutina.

argumentos-SQL
: Corresponde a la lista de parámetros de entrada de la sentencia CREATE de la rutina.

El siguiente es un ejemplo de una función Java que devuelve el producto de sus dos argumentos de entrada:

```
public static double product( double in1, double in2 ) throws SQLException
{
  return in1 * in2;
}
```

La correspondiente sentencia CREATE FUNCTION de esta función escalar es:

```
CREATE FUNCTION product( in1 DOUBLE, in2 DOUBLE )
  RETURNS DOUBLE
  LANGUAGE java
  PARAMETER STYLE java
  NO SQL
  FENCED THREADSAFE
  DETERMINISTIC
  RETURNS NULL ON NULL INPUT
  NO EXTERNAL ACTION
  EXTERNAL NAME 'myjar:udfclass.product'
```

La sentencia anterior presupone que el método es una clase denominada `udfclass` que se encuentra en un archivo JAR que se ha instalado en el servidor de bases de

datos con el ID de Jar `myjar`. Los archivos JAR se pueden instalar en un servidor de bases de datos utilizando el procedimiento definido por el sistema `INSTALL_JAR`.

Rutinas DB2GENERAL

Las rutinas PARAMETER STYLE DB2GENERAL están escritas en Java. La creación de rutinas de DB2GENERAL es muy parecida a la creación de rutinas en otros lenguajes de programación soportados. Una vez que las haya creado y registrado, las podrá llamar desde programas escritos en cualquier lenguaje. Normalmente, puede llamar a las API de JDBC desde los procedimientos almacenados, pero no desde las UDF.

Cuando desarrolle rutinas en Java, es muy recomendable que las registre utilizando la cláusula PARAMETER STYLE JAVA en la sentencia CREATE. PARAMETER STYLE DB2GENERAL sigue disponible para permitir la implementación de las características siguientes en las rutinas Java:

- funciones de tabla
- áreas reutilizables
- acceso a la estructura DBINFO
- la posibilidad de efectuar una FINAL CALL (y una primera llamada separada) a la función o al método

Si tiene rutinas PARAMETER STYLE DB2GENERAL que no utilizan ninguna de las características anteriores, es recomendable migrarlas a PARAMETER STYLE JAVA para su portabilidad.

UDF DB2GENERAL

Puede crear y utilizar UDF en Java tal como lo haría en otros lenguajes, con sólo unas cuantas diferencias menores en comparación con las UDF en C. Después de codificar la UDF, regístrela con la base de datos. Luego podrá consultarla en las aplicaciones.

En general, si declara una UDF que toma argumentos de los tipos de SQL *t1*, *t2* y *t3*, devolviendo el tipo *t4*, se la llamará como método Java con la signatura de Java esperada:

```
public void nombre ( T1 a, T2 b, T3 c, T4 d)  { .....}
```

Donde:

- *nombre* es el nombre del método Java
- De *T1* a *T4* son los tipos de Java que corresponden a los tipos de SQL que van de *t1* a *t4*.
- *a*, *b* y *c* son nombres de variable para los argumentos de entrada.
- *d* es un nombre de variable que representa el argumento de salida.

Por ejemplo, dada una UDF denominada `sample!test3` que devuelve INTEGER y toma argumentos de tipo CHAR(5), BLOB(10K) y DATE, DB2 espera que la implementación en Java de la UDF tenga la signatura siguiente:

```
import COM.ibm.db2.app.*;
public class sample extends UDF {
   public void test3(String arg1, Blob arg2, String arg3,
                     int result) { ... }
}
```

Las rutinas de Java que implementan funciones de tabla requieren más argumentos. Junto a las variables que representan la entrada, en la fila resultante aparece una variable adicional para cada columna. Por ejemplo, una función de tabla se puede declarar como:

```
public void test4(String arg1, int result1,
                  Blob result2, String  result3);
```

Los valores NULL de SQL se representan mediante variables Java que no se inicializan. Estas variables tienen el valor cero si son de tipo primitivo y un valor nulo de Java si son tipos de objeto, en consonancia con las normas de Java. Para indicar un nulo (NULL) de SQL aparte de un cero corriente, puede llamar a la función `isNull` para cualquier argumento de entrada:

```
{ ....
   if (isNull(1)) { /* el argumento #1 era un NULL de SQL */ }
   else           { /* no NULL */ }
}
```

En el ejemplo anterior, los números de argumento empiezan por uno. La función `isNull()`, al igual que las otras funciones que siguen, se hereda de la clase `COM.ibm.db2.app.UDF`.

Para devolver un resultado de una UDF escalar o de tabla, utilice el método `set()` en la UDF, de la manera siguiente:

```
{ ....
   set(2, valor);
}
```

Donde '2' es el índice de un argumento de salida y valor es un literal o una variable de un tipo compatible. El número de argumento es el índice en la lista de argumentos de la salida seleccionada. En el primer ejemplo de este apartado, la variable `int result` tiene el índice 4; en el segundo, de `result1` a `result3` tienen los índices de 2 a 4.

Al igual que sucede con los módulos C utilizados en las UDF y los procedimientos almacenados, no se pueden utilizar las corrientes de E/S estándar Java (`System.in`, `System.out` y `System.err`) en las rutinas Java.

Recuerde que todos los archivos de clases Java (o los JAR que contienen las clases) que utilice para implementar una rutina deben residir en el directorio `sqllib/function` o en un directorio especificado en la variable CLASSPATH del gestor de bases de datos.

Normalmente, DB2 llama a una UDF muchas veces, una vez por cada fila de un conjunto de entrada o de resultado de una consulta. Si se especifica SCRATCHPAD en la sentencia CREATE FUNCTION de la UDF, DB2 reconoce que es necesaria cierta "continuidad" entre invocaciones sucesivas de la UDF y, por consiguiente, no se crea la instancia de implementación de la clase de Java para cada llamada, sino que generalmente se hace así una vez por cada referencia a la UDF por sentencia. Generalmente, se crea una instancia antes de la primera llamada y se utiliza después, pero es posible que se creen instancias de las funciones de tabla con más frecuencia. Sin embargo, si se especifica NO SCRATCHPAD para una UDF, ya sea una función escalar o de tabla, se crea una instancia limpia para cada llamada a la UDF.

Un área reutilizable puede resultar útil para guardar información a lo largo de las llamadas a una UDF. Mientras que las UDF de Java y OLE pueden utilizar

variables de instancias o establecer el área reutilizable para lograr una continuidad entre llamadas, las UDF de C y C++ deben utilizar el área reutilizable. Las UDF de Java acceden al área reutilizable mediante los métodos getScratchPad() y setScratchPad() disponibles en `COM.ibm.db2.app.UDF`.

Para las funciones de tabla de Java que utilicen un área reutilizable, controle cuándo se debe obtener una nueva instancia del área reutilizable mediante las opciones FINAL CALL o NO FINAL CALL de la sentencia CREATE FUNCTION.

La posibilidad de lograr una continuidad entre llamadas a una UDF por medio de un área reutilizable se controla mediante las opciones SCRATCHPAD y NO SCRATCHPAD de CREATE FUNCTION, independientemente de si se utilizan el área reutilizable de DB2 o variables de instancia.

Para las funciones escalares, se utiliza la misma instancia para la sentencia entera.

Observe que cada referencia a una UDF de Java en una consulta se trata de forma independiente, aunque se haga referencia a la misma UDF varias veces. Esto es lo mismo que sucede con las UDF de OLE, C y C++. Al final de una consulta, si se especifica la opción FINAL CALL para una función escalar, se llama al método close() del objeto. Para las funciones de tabla, siempre se invocará el método close(), tal como se indica en el subapartado que sigue a éste. Si no se define un método close() para la clase UDF, una función stub toma el control y se ignora el suceso.

Si se especifica la cláusula ALLOW PARALLEL para una UDF de Java en la sentencia CREATE FUNCTION, DB2 puede elegir evaluar la UDF en paralelo. Si así sucede, se pueden crear varios objetos Java diferenciados en distintas particiones. Cada objeto recibe un subconjunto de las filas.

Al igual que sucede con otras UDF, las UDF de Java pueden ser protegidas (FENCED) o no protegidas (NOT FENCED). Las UDF NOT FENCED se ejecutan dentro del espacio de direcciones del mecanismo de bases de datos; las UDF FENCED se ejecutan en un proceso separado. Aunque las UDF de Java no pueden corromper inadvertidamente el espacio de direcciones del proceso que las incorpora, pueden interrumpir o ralentizar el proceso. Por consiguiente, cuando depure UDF escritas en Java, las debe ejecutar como UDF FENCED.

Tipos de datos de SQL soportados en rutinas DB2GENERAL

Cuando se llama a rutinas con PARAMETER STYLE DB2GENERAL, DB2 convierte los tipos de SQL en tipos de Java, y viceversa, en nombre del usuario. En el paquete de Java, `COM.ibm.db2.app` se proporcionan varias de estas clases.

Tabla 41. Tipos de SQL y objetos Java de DB2

Tipo de columna SQL	Tipo de datos de Java
SMALLINT	short
INTEGER	int
BIGINT	long
REAL[1]	float
DOUBLE	double
DECIMAL(p,s)	java.math.BigDecimal
NUMERIC(p,s)	java.math.BigDecimal

Tabla 41. Tipos de SQL y objetos Java de DB2 (continuación)

Tipo de columna SQL	Tipo de datos de Java
CHAR(*n*)	java.lang.String
CHAR(*n*) FOR BIT DATA	COM.ibm.db2.app.Blob
VARCHAR(*n*)	java.lang.String
VARCHAR(*n*) FOR BIT DATA	COM.ibm.db2.app.Blob
LONG VARCHAR	java.lang.String
LONG VARCHAR FOR BIT DATA	COM.ibm.db2.app.Blob
GRAPHIC(*n*)	java.lang.String
VARGRAPHIC(*n*)	String
LONG VARGRAPHIC[2]	String
BLOB(*n*)[2]	COM.ibm.db2.app.Blob
CLOB(*n*)[2]	COM.ibm.db2.app.Clob
DBCLOB(*n*)[2]	COM.ibm.db2.app.Clob
DATE[3]	String
TIME[3]	String
TIMESTAMP[3]	String

Nota:

1. La diferencia entre REAL y DOUBLE en la SQLDA es el valor de la longitud (4 u 8).
2. Las clases Blob y Clob se proporcionan en el paquete `COM.ibm.db2.app`. Sus interfaces incluyen rutinas para generar una Corriente de entrada y una Corriente de salida para leer y grabar en un Blob, y un Lector y Grabador para un Clob.
3. Los valores de DATE, TIME y TIMESTAMP de SQL utilizan la codificación de series ISO en Java, al igual que para las UDF codificadas en C.
4. Se sigue ofreciendo soporte para los tipos de datos LONG VARCHAR y LONG VARGRAPHIC, pero éstos se han dejado de utilizar y no se recomiendan ya que puede que se eliminen en un release futuro.

Las instancias de las clases `COM.ibm.db2.app.Blob` y `COM.ibm.db2.app.Clob` representan los tipos de datos LOB (BLOB, CLOB y DBCLOB). Estas clases proporcionan una interfaz limitada para los LOB de lectura pasados como entrada y los LOB de grabación devueltos como salida. La lectura y la grabación de los LOB se producen a través de objetos estándar de corrientes de E/S de Java. Para la clase Blob, las rutinas `getInputStream()` y `getOutputStream()` devuelven un objeto de Corriente de entrada o de Corriente de salida mediante el cual se puede procesar el contenido del BLOB, unos cuantos bytes cada vez. Para un Clob, las rutinas `getReader()` y getWriter() devolverán un objeto Lector o Grabador mediante el cual se puede procesar el contenido del CLOB o del DBCLOB, unos cuantos caracteres cada vez.

Si se devuelve un objeto de este tipo como salida utilizando el método `set()`, se pueden aplicar conversiones de página de códigos para representar los caracteres Unicode de Java en la página de códigos de la base de datos.

Clases de Java para rutinas DB2GENERAL

Esta interfaz proporciona la rutina siguiente para captar una conexión JDBC con el contexto de la aplicación que realiza la incorporación:

```
public java.sql.Connection getConnection()
```

Puede utilizar este descriptor de contexto para ejecutar sentencias de SQL. En el archivo `sqllib/samples/java/StoredProc.java` se listan otros métodos de la interfaz `StoredProc`.

Existen cinco clases/interfaces que se pueden utilizar con Procedimientos almacenados o UDF de Java:

- COM.ibm.db2.app.StoredProc
- COM.ibm.db2.app.UDF
- COM.ibm.db2.app.Lob
- COM.ibm.db2.app.Blob
- COM.ibm.db2.app.Clob

Clase DB2GENERAL de Java: COM.IBM.db2.app.StoredProc

Una clase de Java que contiene métodos pensados para que se llamen como procedimientos almacenados PARAMETER STYLE DB2GENERAL debe ser pública y debe implementar esta interfaz de Java. Debe declarar una clase de este tipo del modo siguiente:

```
public class clase-STP-usuario extends COM.ibm.db2.app.StoredProc{ ... }
```

Sólo puede llamar a métodos heredados de la interfaz `COM.ibm.db2.app.StoredProc` en el contexto del procedimiento almacenado que se está ejecutando actualmente. Por ejemplo, no se pueden utilizar operaciones sobre argumentos LOB ni llamadas de establecimiento de resultados o estados después de que se devuelva un procedimiento almacenado. De violar esta norma, se emitirá una excepción de Java.

Las llamadas relacionadas con argumentos utilizan un índice de columnas para identificar la columna a la que se hace referencia. Empiezan por 1 para el primer argumento. Todos los argumentos de un procedimiento almacenado PARAMETER STYLE DB2GENERAL se consideran INOUT y, por lo tanto, constituyen tanto entradas como salidas.

La base de datos capta cualquier excepción devuelta desde el procedimiento almacenado y la devuelve al llamador con SQLCODE -4302, SQLSTATE 38501. Una SQLException o SQLWarning de JDBC se maneja de forma especial y pasa literalmente sus propios SQLCODE, SQLSTATE, etc., a la aplicación llamadora.

Los métodos siguientes están asociados a la clase `COM.ibm.db2.app.StoredProc`:

```
public StoredProc() [constructor por omisión]
```

La base de datos llama a este constructor antes de la llamada al procedimiento almacenado.

```
public boolean isNull(int) throws Exception
```

Esta función prueba si un argumento de entrada que tiene el índice indicado es un nulo (NULL) de SQL.

```
public void set(int, short) throws Exception
public void set(int, int) throws Exception
public void set(int, double) throws Exception
public void set(int, float) throws Exception
public void set(int, java.math.BigDecimal) throws Exception
public void set(int, String) throws Exception
public void set(int, COM.ibm.db2.app.Blob) throws Exception
public void set(int, COM.ibm.db2.app.Clob) throws Exception
```

Esta función establece el argumento de salida que tiene el índice indicado con el valor proporcionado. El índice tiene que hacer referencia a un argumento de salida válido, el tipo de datos debe coincidir y el valor debe tener una longitud y un contenido aceptables. Las series que contienen caracteres Unicode se tienen que poder representar en la página de códigos de la base de datos. Si se producen errores, se emite una excepción.

```
public java.sql.Connection getConnection() throws Exception
```

Esta función devuelve un objeto JDBC que representa la conexión de la aplicación llamadora con la base de datos. Es análogo al resultado de una llamada `SQLConnect()` nula en un procedimiento almacenado C.

Clase DB2GENERAL de Java: COM.IBM.db2.app.UDF

Una clase de Java que contiene métodos pensados para que se llamen como UDF PARAMETER STYLE DB2GENERAL debe ser pública y debe implementar esta interfaz de Java. Debe declarar una clase de este tipo del modo siguiente:

```
public class clase-UDF-usuario extends COM.ibm.db2.app.UDF{ ... }
```

Sólo se puede llamar a métodos de la interfaz `COM.ibm.db2.app.UDF` en el contexto de la UDF que se esté ejecutando actualmente. Por ejemplo, no se pueden utilizar operaciones sobre argumentos LOB, llamadas de establecimiento de resultados o estados, etc., después de que una UDF vuelve. De violar esta norma, se emitirá una excepción de Java.

Las llamadas relacionadas con argumentos utilizan un índice de columnas para identificar la columna que se está estableciendo. Empiezan por 1 para el primer argumento. Los argumentos de salida se numeran con números más altos que los de entrada. Por ejemplo, una UDF escalar con tres entradas utiliza el índice 4 para la salida.

La base de datos capta cualquier excepción devuelta por la UDF y la devuelve al llamador con SQLCODE -4302, SQLSTATE 38501.

Los métodos siguientes están asociados a la clase `COM.ibm.db2.app.UDF`:

```
public UDF() [constructor por omisión]
```

La base de datos llama a este constructor al principio de una serie de llamadas a UDF. Precede a la primera llamada a la UDF.

```
public void close()
```

La base de datos llama a esta función al final de la evaluación de una UDF, en caso de que la UDF se haya creado con la opción FINAL CALL. Es análogo a la llamada final para una UDF en C. Para las funciones de tabla, se llama a close() después de la llamada CLOSE al método UDF (si se ha codificado o se ha asumido por omisión NO FINAL CALL), o después de la llamada FINAL (si se ha codificado FINAL CALL). Si una clase UDF de Java no implementa esta función, un apéndice no-op manejará e ignorará este suceso.

```
public int getCallType() throws Exception
```

Los métodos UDF de funciones de tabla utilizan getCallType() para averiguar el tipo de llamada de una llamada determinada. Devuelve un valor como los siguientes (se proporcionan definiciones simbólicas para estos valores en la definición de la clase COM.ibm.db2.app.UDF):

- -2 FIRST call

- -1 OPEN call
- 0 FETCH call
- 1 CLOSE call
- 2 FINAL call

```
public boolean isNull(int) throws Exception
```

Esta función prueba si un argumento de entrada que tiene el índice indicado es un nulo (NULL) de SQL.

```
public boolean needToSet(int) throws Exception
```

Esta función prueba si es necesario establecer un argumento de salida con el índice indicado. Esto puede ser falso para una UDF de tabla declarada con DBINFO, en caso de que el llamador de la UDF no utilice esa columna.

```
public void set(int, short) throws Exception
public void set(int, int) throws Exception
public void set(int, double) throws Exception
public void set(int, float) throws Exception
public void set(int, java.math.BigDecimal) throws Exception
public void set(int, String) throws Exception
public void set(int, COM.ibm.db2.app.Blob) throws Exception
public void set(int, COM.ibm.db2.app.Clob) throws Exception
```

Esta función establece el argumento de salida que tiene el índice indicado con el valor proporcionado. El índice tiene que hacer referencia a un argumento de salida válido, el tipo de datos debe coincidir y el valor debe tener una longitud y un contenido aceptables. Las series que contienen caracteres Unicode se tienen que poder representar en la página de códigos de la base de datos. Si se producen errores, se emite una excepción.

```
public void setSQLstate(String) throws Exception
```

Se puede llamar a esta función desde una UDF para establecer el SQLSTATE que esta llamada devolverá. Una UDF de tabla debe llamar a esta función con "02000" para indicar una condición de fin de tabla. Si la serie no es aceptable como SQLSTATE, se emitirá una excepción.

```
public void setSQLmessage(String) throws Exception
```

Esta función es parecida a la función `setSQLstate`. Establece el resultado del mensaje de SQL. Si la serie no es aceptable (por ejemplo, si contiene más de 70 caracteres), se emitirá una excepción.

```
public String getFunctionName() throws Exception
```

Esta función devuelve el nombre de la UDF que se está ejecutando.

```
public String getSpecificName() throws Exception
```

Esta función devuelve el nombre específico de la UDF que se está ejecutando.

```
public byte[] getDBinfo() throws Exception
```

Esta función devuelve una estructura DBINFO pura no procesada para la UDF que se está ejecutando, como matriz de bytes. En primer lugar, se debe declarar con la opción DBINFO.

```
public String getDBname() throws Exception
public String getDBauthid() throws Exception
public String getDBtbschema() throws Exception
public String getDBtbname() throws Exception
public String getDBcolname() throws Exception
```

```
public String getDBver_rel() throws Exception
public String getDBplatform() throws Exception
public String getDBapplid() throws Exception
```

Estas funciones devuelven el valor del campo apropiado de la estructura DBINFO de la UDF que se está ejecutando.

```
public int getDBprocid() throws Exception
```

Esta función devuelve el id de rutina del procedimiento que ha invocado, directa o indirectamente, a esta rutina. El id de rutina coincide con la columna ROUTINEID de SYSCAT.ROUTINES que se puede utilizar para recuperar el nombre del procedimiento que realiza la invocación. Si se invoca la rutina que se ejecuta desde una aplicación, getDBprocid() devuelve 0.

```
public int[] getDBcodepg() throws Exception
```

Esta función devuelve el SBCS, el DBCS y los números de página de códigos compuestos para la base de datos, desde la estructura DBINFO. La matriz de enteros devuelta contiene los números respectivos como sus tres primeros elementos.

```
public byte[] getScratchpad() throws Exception
```

Esta función devuelve una copia del cuaderno de apuntes de la UDF que se está ejecutando actualmente. En primer lugar, se debe declarar la UDF con la opción SCRATCHPAD.

```
public void setScratchpad(byte[]) throws Exception
```

Esta función sobregraba el cuaderno de apuntes de la UDF que se está ejecutando actualmente con el contenido de la matriz de bytes indicada. En primer lugar, se debe declarar la UDF con la opción SCRATCHPAD. La matriz de bytes debe tener el mismo tamaño que `getScratchpad()` devuelve.

Clase DB2GENERAL de Java: COM.IBM.db2.app.Lob

Esta clase proporciona rutinas de utilidad que crean objetos Blob o Clob temporales para un cálculo interno en las rutinas.

Los métodos siguientes están asociados a la clase COM.ibm.db2.app.Lob:

```
public static Blob newBlob() throws Exception
```

Esta función crea un Blob temporal. Si es posible, se implementará utilizando un LOCALIZADOR.

```
public static Clob newClob() throws Exception
```

Esta función crea un Clob temporal. Si es posible, se implementará utilizando un LOCALIZADOR.

Clase DB2GENERAL de Java: COM.IBM.db2.app.Blob

La base de datos pasa una instancia de esta clase para representar un BLOB como entrada de la rutina, y se puede devolver como salida. La aplicación puede crear instancias, pero sólo en el contexto de una rutina en ejecución. Los usos de estos objetos fuera de este tipo de contexto emitirán una excepción.

Los métodos siguientes están asociados a la clase `COM.ibm.db2.app.Blob`:

```
public long size() throws Exception
```

Esta función devuelve la longitud (en bytes) del BLOB.

```
public java.io.InputStream getInputStream() throws Exception
```

Esta función devuelve una nueva Corriente de entrada para leer el contenido del BLOB. Se dispone de operaciones eficaces de búsqueda/marcaje sobre este objeto.

```
public java.io.OutputStream getOutputStream() throws Exception
```

Esta función devuelve una nueva Corriente de salida para añadir bytes al BLOB. Los bytes añadidos pasan a ser visibles de inmediato en todas las instancias de Corriente de entrada existentes producidas por la llamada `getInputStream()` de este objeto.

Clase DB2GENERAL de Java: COM.IBM.db2.app.Clob

La base de datos pasa una instancia de esta clase para representar un CLOB o DBCLOB como entrada de la rutina, y se puede devolver como salida. La aplicación puede crear instancias, pero sólo en el contexto de una rutina en ejecución. Los usos de estos objetos fuera de este tipo de contexto emitirán una excepción.

Las instancias de Clob almacenan caracteres en la página de códigos de la base de datos. Algunos caracteres Unicode no se pueden representar en esta página de códigos y pueden causar que se emita una excepción durante la conversión. Esto se puede producir durante una operación de adición o durante una llamada `set()` de una UDF o de StoredProc. Es necesario para ocultar la distinción entre un CLOB y un DBCLOB del programador Java.

Los métodos siguientes están asociados a la clase `COM.ibm.db2.app.Clob`:

```
public long size() throws Exception
```

Esta función devuelve la longitud (en caracteres) del CLOB.

```
public java.io.Reader getReader() throws Exception
```

Esta función devuelve un nuevo Lector para leer el contenido del CLOB o DBCLOB. Se dispone de operaciones eficaces de búsqueda/marcaje sobre este objeto.

```
public java.io.Writer getWriter() throws Exception
```

Esta función devuelve un nuevo Grabador para añadir caracteres a este CLOB o DBCLOB. Los caracteres añadidos pasan a ser visibles de inmediato en todas las instancias de Lector existentes producidas por la llamada `GetReader()` de este objeto.

Paso de parámetros de tipo ARRAY a rutinas Java

En DB2 9.5. se da soporte a la posibilidad de pasar y obtener parámetros del tipo ARRAY con procedimientos Java.

Acerca de esta tarea

Debería elegir la implementación de la utilización del paso de matrices entre aplicaciones y procedimientos almacenados Java si desea pasar:

- Un volumen grande de datos homogéneos a un procedimiento mediante un único parámetro.

- Un número variable de entradas del mismo tipo de datos a un procedimiento utilizando un único parámetro.

Por ejemplo, se podrían pasar todos los nombres de los estudiantes de una clase a un procedimiento utilizando un único parámetro sin conocer el número de estudiantes. Si no se utilizase un tipo de datos ARRAY, se necesitaría un parámetro por cada nombre de estudiante.

Procedimiento

Si desea pasar un parámetro de tipo ARRAY:

1. El tipo de datos ARRAY ya debe estar definido. Si desea definir un tipo de matriz, se debe ejecutar una sentencia CREATE TYPE.
2. La definición del procedimiento debe incluir un parámetro del tipo definido. Por ejemplo, si se crea si un tipo de datos ARRAY denominado `IntArray` ejecutando una sentencia CREATE TYPE, haga lo siguiente para pasar un parámetro de este tipo al procedimiento.

Ejemplo

```
CREATE PROCEDURE inArray (IN input IntArray)
LANGUAGE JAVA
PARAMETER STYLE JAVA
EXTERNAL NAME 'MyProcs:MyArrayProcs!inArray';
```

En la definición del procedimiento, el parámetro de matriz se especifica como java.sql.Array. Dentro del procedimiento, se correlaciona el argumento con una matriz Java utilizando el método getArray(), tal como se muestra a continuación. Observe la utilización de Integer en lugar de int (u otros tipos de primitivas) con las matrices.

```
static void inArray(java.sql.Array input)
{
Integer[] inputArr = (Integer [])input.getArray();
int sum = 0;
for(int i=0, i < inputArr.length; i++)
{
sum += inputArr[i];
}
}
```

Devolución de conjuntos de resultados desde procedimientos JDBC

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

Para devolver un conjunto de resultados de un procedimiento JDBC:

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

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

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

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

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

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

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

4. Ejecute la sentencia:

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

5. Finalice el cuerpo del procedimiento.

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

Devolución de conjuntos de resultados desde procedimientos de SQLJ

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

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

1. Declare una clase de iterador para manejar datos de consulta. Por ejemplo:

```
#sql iterator SpServerEmployees(String, String, double);
```

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

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

3. Cree una instancia de un objeto de iterador. Por ejemplo:

```
SpServerEmployees c1;
```

4. Asigne la sentencia de SQL que generará el conjunto de resultados a un iterador. En el ejemplo siguiente, se utiliza una variable del lenguaje principal (denominada inSalaryThreshold -- vea el ejemplo de signatura de función anterior) en la cláusula WHERE de la consulta:

```
#sql c1 = {SELECT name, job, CAST(salary AS DOUBLE)
             FROM staff
             WHERE salary > :inSalaryThreshold
             ORDER BY salary};
```

5. Ejecute la sentencia y obtenga el conjunto de resultados:

```
rs[0] = c1.getResultSet();
```

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

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

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

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

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

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

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

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

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

3. Llame al procedimiento:

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

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

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

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

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

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

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

  // Las filas del conjunto de resultados se captan y se imprimen en pantalla.
  while (rs.next())
```

```
  {
    r++;
    System.out.print("Row: " + r + ": ");
    for (int i=1; i <= numOfColumns; i++)
    {
      System.out.print(rs.getString(i));
      if (i != numOfColumns)
      {
        System.out.print(", ");
      }
    }
    System.out.println();
  }
}
```

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

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

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

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

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

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

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

2. Establezca el contexto por omisión (utilizando un objeto DefaultContext):

```
DefaultContext ctx = new DefaultContext(con);
DefaultContext.setDefaultContext(ctx);
```

3. Establezca el contexto de ejecución (utilizando un objeto ExecutionContext):

```
ExecutionContext execCtx = ctx.getExecutionContext();
```

4. Invoque un procedimiento que devuelva conjuntos de resultados. En el ejemplo siguiente, se invoca un procedimiento denominado GET_HIGH_SALARIES, y se pasa una variable de entrada (llamada inSalaryThreshold):

```
#sql {CALL GET_HIGH_SALARIES(:in inSalaryThreshold, :out outErrorCode)};
```

5. Declare un objeto ResultSet y utilice el método getNextResultSet() del objeto ExecutionContext a fin de aceptar conjuntos de resultados del procedimiento. Si son varios conjuntos de resultados, coloque la llamada getNextResultSet() en una estructura de bucle. Cada conjunto de resultados devuelto por el procedimiento generará una iteración de bucle. Dentro del bucle, puede captar las filas del conjunto de resultados con su método y luego cerrar el objeto de conjunto de resultados (con el método close() del objeto ResultSet). Por ejemplo:

```
ResultSet rs = null;

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

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

```
        System.out.print(rs.getString(i));
        if (i != numOfColumns)
        {
          System.out.print(", ");
        }
      }
      System.out.println();
    }

    rs.close();
  }
```

Restricciones de las rutinas Java

Las siguientes restricciones se aplican a las rutinas Java:

- El procedimiento definido por el sistema `install_jar` que se utiliza para desplegar el código de rutina Java en archivos JAR para el sistema de archivos del servidor de bases de datos no se puede invocar en una aplicación de base de datos Java cuando se utiliza el controlador de DB2 Universal JDBC. Este controlador no da soporte a este procedimiento.

  La alternativa recomendad es utilizar el Procesador de línea de mandatos de DB2.
- La cláusula PROGRAM TYPE MAIN no está soportada en sentencias CREATE PROCEDURE o CREATE FUNCTION para rutinas Java independientemente del valor de cláusula PARAMETER STYLE especificado.
- Las siguientes características no están soportadas en el estilo de parámetro JAVA:
  - funciones de tabla
  - scratchpads en funciones
  - acceso a la estructura DBINFO en funciones
  - la invocación de FINAL CALL en funciones

  La alternativa recomendada si necesita estas funciones es crear una función Java utilizando el estilo de parámetro DB2GENERAL o crear la función utilizando el lenguaje de programación C ó C++ .
- Las llamadas de Java Native Interface (JNI) desde las rutinas Java no están soportadas.

  Si es necesario invocar código C ó C++ desde una rutina Java, puede hacerlo invocando una rutina C ó C++ definida por separado.
- Actualmente las rutinas Java NOT FENCED no reciben soporte. Una rutina Java definida como NOT FENCED se invocará como si se hubiera definido como FENCED THREADSAFE.
- Los procedimientos almacenados de Java no pueden depender de cualquier recurso que no sea del sistema, como por ejemplo los archivos de propiedades. Si llama a un procedimiento almacenado de Java que dependa de un recurso que no sea del sistema, los recursos no estarán cargados y no se devolverá ningún error.

Modelo de ejecución de las funciones para Java

Para las funciones de tabla escritas en Java que utilizan PARAMETER STYLE DB2GENERAL, es importante comprender lo que sucede en cada punto del proceso de DB2 de una sentencia determinada. La tabla siguiente detalla esta

información para una función de tabla habitual. Se abarcan los casos de NO FINAL CALL y de FINAL CALL, suponiendo en ambos casos SCRATCHPAD.

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

Nota:

1. El término "método de UDF" hace referencia al método de clase de Java que implementa la UDF. Se trata del método identificado en la cláusula EXTERNAL NAME de la sentencia CREATE FUNCTION.
2. Para las funciones de tabla que tengan especificado NO SCRATCHPAD, las llamadas al método de UDF son las indicadas en esta tabla, pero, puesto que el usuario no solicita ninguna continuidad por medio de un área reutilizable, DB2 hará que se cree una instancia de un nuevo objeto antes de cada llamada,

llamando al constructor de clases. No está claro que las funciones de tabla con NO SCRATCHPAD (y, por lo tanto, sin continuidad) puedan realizar acciones útiles, pero se soportan.

Creación de rutinas Java

La creación de rutinas Java consiste en:

- Ejecutar una sentencia CREATE que define la rutina en un servidor de bases de datos de DB2
- Desarrollar la implementación de la rutina que se corresponde con la definición de la rutina.

A continuación, se citan las formas en las que se pueden crear rutinas Java:

- Utilización de IBM Data Studio
- Utilizar las características de desarrollo de rutinas de DB2 en IBM Rational Application Developer
- Utilizar la ventana de mandatos de DB2

En general resulta más sencillo crear rutinas Java con el IBM Data Studio, aunque muchos desarrolladores tienen la posibilidad de crear rutinas Java desde el entorno de desarrollo Java integrado que facilita IBM Rational Application Developer. Si estas herramientas gráficas no se pueden utilizar, la ventana de mandatos de DB2 proporciona un soporte similar por medio de una interfaz de línea de mandatos.

Requisitos previos

- Revise el Capítulo 8, “Rutinas Java”, en la página 303.
- Asegúrese de que tiene acceso a un servidor de DB2 Versión 9, incluyendo instancias y bases de datos.
- Asegúrese de que el sistema operativo está en un nivel de versión soportado por los productos de base de datos DB2.
- Asegúrese de que el “Software de desarrollo de rutinas Java soportado” en la página 304 esté en un nivel de versión que esté soportado para el desarrollo de las rutinas Java.
- Asegúrese de que el desarrollo de la “Especificación de un controlador para rutinas Java” en la página 306 sea válido.
- Autorización para ejecutar la sentencia CREATE PROCEDURE o CREATE FUNCTION.

Para obtener una lista de las restricciones asociadas a rutinas Java, consulte:

- “Restricciones de las rutinas Java” en la página 326

Cree rutinas Java utilizando uno de los procedimientos siguientes:

- Crear rutinas Java mediante IBM Data Studio
- Crear rutinas Java mediante Rational Application Developer
- “Creación de rutinas Java desde la línea de mandatos”

Creación de rutinas Java desde la línea de mandatos

Los procedimientos y funciones que hacen referencia a una clase Java se crean de forma similar a las rutinas externas con otras implementaciones. Esta tarea comprende unos pasos, incluida la formulación de la sentencia CREATE para la

rutina, la codificación y compilación (conversión) de la implementación de la rutina y el despliegue de la clase Java en un servidor de bases de datos DB2.

Elegirá implementar una rutina Java si:

- Desea encapsular la lógica compleja en una rutina que acceda a la base de datos o que realice una acción fuera de la base de datos.
- Necesita que la lógica encapsulada se invoque desde cualquiera de estos elementos: diversas aplicaciones, el CLP, otra rutina (procedimiento, función (UDF) o método) o un activador.
- Se siente más cómodo al codificar esta lógica utilizando Java y una de las interfaces de programación de aplicaciones JDBC o SQLJ.

Requisitos previos

- Revise el Capítulo 8, "Rutinas Java", en la página 303.
- Asegúrese de que tiene acceso a un servidor de bases de datos DB2 Versión 9, incluyendo instancias y bases de datos.
- Asegúrese de que el sistema operativo está en un nivel de versión soportado por los productos de base de datos de DB2.
- Asegúrese de que el "Software de desarrollo de rutinas Java soportado" en la página 304 esté en un nivel de versión que esté soportado para el desarrollo de las rutinas Java.
- Asegúrese de que el desarrollo de la "Especificación de un controlador para rutinas Java" en la página 306 sea válido.
- Autorización para ejecutar la sentencia CREATE PROCEDURE o CREATE FUNCTION.

Procedimiento

1. Codifique la lógica de la rutina en Java.
   - Para obtener información general sobre las rutinas Java y las características de las rutinas Java, consulte los temas a los que se hace referencia en la sección Requisitos previos.
   - Debe implementarse una signatura de parámetro de rutina utilizando uno de los estilos de parámetros soportados. Se recomienda muy especialmente utilizar el estilo de parámetro JAVA para todas las rutinas Java. Si desea más información sobre las signaturas de parámetros y las implementaciones de parámetros, consulte:
     - "Parámetros de las rutinas Java" en la página 310
     - "Estilo de parámetro en procedimientos JAVA" en la página 310
     - "Funciones y métodos Java con parámetros de estilo JAVA" en la página 312
   - Declare variables de la misma manera que lo hace para las aplicaciones de bases de datos Java. Utilice correctamente los tipos de datos que se correlacionan con tipos de datos de SQL de DB2. Para obtener más información sobre la correlación de tipos de datos entre tipos de datos DB2 y Java, consulte:
     - "Tipos de datos que se correlacionan con tipos de datos de base de datos en aplicaciones Java" en *Desarrollo de aplicaciones Java*
   - Incluya la lógica de la rutina. La lógica de la rutina puede consistir en cualquier código soportado en el lenguaje de programación Java. También puede incluir la ejecución de sentencias de SQL de la misma manera que se

hace para las aplicaciones de base de datos Java. Para obtener más información sobre la ejecución de sentencias SQL en el código de Java, consulte:

– "Interfaces JDBC para ejecutar SQL" en *Desarrollo de aplicaciones Java*

– "Ejecución de sentencias de SQL en aplicaciones SQLJ" en *Desarrollo de aplicaciones Java*

- Si la rutina es un procedimiento y desea devolver un conjunto de resultados al llamador de la rutina, no es necesario ningún parámetro para el conjunto de resultados. Para obtener más información sobre la devolución de conjuntos de resultados de las rutinas Java:

  – "Devolución de conjuntos de resultados desde procedimientos JDBC" en la página 322

  – "Devolución de conjuntos de resultados desde procedimientos de SQLJ" en la página 323

- Establezca un valor de retorno de rutina al final de la rutina.

2. Cree el código para generar un archivo de clase Java o un archivo JAR que contenga un conjunto de archivos de clase Java. Para obtener información sobre cómo crear código de rutina Java, consulte:
   - "Creación de rutinas JDBC" en *Desarrollo de aplicaciones Java*
   - "Creación de rutinas JDBC" en *Desarrollo de aplicaciones Java*
3. Copie el archivo de clase en el servidor de bases de datos de DB2 o instale el archivo JAR en el servidor de bases de datos de DB2. Para obtener más información sobre cómo efectuar esta acción, consulte:
   - "Despliegue de archivos de clases de rutinas Java en servidores de base de datos DB2" en la página 334
   - "Administración de los archivos JAR en el servidor de bases de datos" en la página 335

   Es recomendable almacenar los archivos de clases asociados con rutinas de DB2 en el *directorio de función*. Para conocer más acerca del directorio de función, consulte la información relacionada con la cláusula `EXTERNAL` de una de las sentencias siguientes: `CREATE PROCEDURE` o `CREATE FUNCTION`.

   Puede copiar la biblioteca en otro directorio del servidor si lo desea, pero, para invocar satisfactoriamente la rutina, debe anotar el nombre de vía de acceso completamente calificado de la biblioteca porque lo necesitará en el paso siguiente.
4. Ejecute de forma dinámica o estática la sentencia CREATE correspondiente para el tipo de rutina: `CREATE PROCEDURE` o `CREATE FUNCTION` .
   - Especifique la cláusula `LANGUAGE` con: `JAVA`
   - Especifique la cláusula `PARAMETER STYLE` con el nombre del estilo de parámetro soportado que se ha implementado en el código de la rutina. Se recomienda utilizar `PARAMETER STYLE JAVA` a menos que las características que necesite sólo estén soportadas al utilizar PARAMETER STYLE DB2GENERAL.
   - Especifique la cláusula `EXTERNAL` con el nombre del archivo JAR o la clase Java que se ha de asociar con la rutina utilizando uno de los valores siguientes:

     – el nombre de vía de acceso completamente calificado del archivo de clase Java

     – el nombre de vía de acceso relativo del archivo de clase Java relativo al directorio de función.

– el ID de archivo JAR del archivo JAR del servidor de bases de datos que contiene la clase Java

Por omisión DB2 buscará la biblioteca en el directorio de función, a menos que se especifique una clase e ID de archivo JAR, un nombre de vía de acceso completamente calificado o relativo para la biblioteca en la cláusula EXTERNAL.

- Especifique `DYNAMIC RESULT SETS` con un valor numérico si la rutina es un procedimiento y ha de devolver uno o más conjuntos de resultados al llamador.
- Especifique otros valores de cláusula no por omisión en la sentencia CREATE a utilizar para caracterizar la rutina.

Para invocar la rutina Java, consulte el Capítulo 11, "Invocación de rutinas", en la página 355

Creación del código de rutinas Java

Una vez se ha escrito el código de implementación de rutinas Java, se debe crear antes de que el ensamblaje de la rutina se pueda desplegar y de que se pueda invocar la rutina. Los pasos necesarios para crear rutinas Java son similares a los que se necesitan para crear cualquier rutina externa, aunque hay algunas diferencias.

Hay varias formas de crear rutinas Java:

- Mediante las herramientas gráficas que se proporcionan con IBM Data Studio
- Mediante las herramientas gráficas facilitadas en IBM Rational Application Developer
- Mediante scripts de creación de ejemplos de DB2
- Entrando mandatos desde una ventana de mandatos de DB2

Las herramientas gráficas y DB2 se pueden personalizar para crear rutinas Java para diversos sistemas operativos y con diversos valores. Los scripts de creación de ejemplo y los archivos de proceso por lotes correspondientes a rutinas están diseñados para crear rutinas de ejemplo de DB2 (procedimientos y funciones definidas por el usuario) para un determinado sistema operativo utilizando el software de desarrollo soportado por omisión.
Hay un conjunto independiente de scripts de creación de ejemplo de DB2 y de archivos de proceso por lotes para las rutinas Java creadas con JDBC y SQLJ. En general, es más fácil crear rutinas Java utilizando las herramientas gráficas o los scripts de creación que se pueden modificar fácilmente, si hace falta, aunque a menudo resulta útil saber también cómo se crean rutinas desde las ventanas de mandatos de DB2.

Creación de rutinas JDBC

Puede utilizar un archivo `makefile` de Java o el mandato javac para crear rutinas JDBC. Una vez creadas esas rutinas, es necesario catalogarlas.

Los pasos siguientes muestran cómo crear y ejecutar estas rutinas:

- El procedimiento almacenado de ejemplo `SpServer` de JDBC
- La función definida por el usuario de ejemplo `UDFsrv`, que no tiene ninguna sentencia de SQL
- La función definida por el usuario de ejemplo `UDFsqlsv`, que tiene sentencias de SQL

- Para crear y ejecutar el procedimiento almacenado SpServer.java en el servidor, desde la línea de mandatos:
  1. Compile SpServer.java para crear el archivo SpServer.class, mediante este mandato:
     ```
     javac SpServer.java
     ```
  2. Copie `SpServer.class` en el directorio `sqllib\function` (en sistemas operativos Windows) o en el directorio `sqllib/function` (en UNIX).
  3. Catalogue las rutinas ejecutando el script spcat en el servidor. El script spcat conecta con la base de datos de ejemplo, descataloga las rutinas si fueron catalogadas previamente mediante SpDrop.db2, luego las cataloga utilizando SpCreate.db2, y finalmente desconecta de la base de datos. También puede ejecutar los scripts SpDrop.db2 y SpCreate.db2 por separado.
  4. Detenga y reinicie la base de datos para que se pueda reconocer el nuevo archivo de clase. Si es necesario, defina la modalidad de archivo del archivo de clase como "read" para que pueda ser leído por el usuario.
  5. Compile y ejecute la aplicación cliente `SpClient` para acceder a la clase del procedimiento almacenado.
- Para crear y ejecutar el programa de la función definida por el usuario `UDFsrv.java` (función definida por el usuario sin ninguna sentencia de SQL) en el servidor, realice lo siguiente desde la línea de mandatos:
  1. Compile `UDFsrv.java` para crear el archivo UDFsrv.class, mediante este mandato:
     ```
     javac UDFsrv.java
     ```
  2. Copie `UDFsrv.class` en el directorio `sqllib\function` (en sistemas operativos Windows) o en el directorio `sqllib/function` (en UNIX).
  3. Compile y ejecute un programa cliente por el que se invoque UDFsrv.

     Para acceder a la biblioteca de `UDFsrv`, puede utilizar la aplicación UDFcli.java de JDBC o la aplicación cliente UDFcli.sqlj de SQLJ. Ambas versiones del programa cliente contienen la sentencia CREATE FUNCTION de SQL, que permite registrar las funciones definidas por el usuario en la base de datos, y también contiene sentencias de SQL que hacen uso de funciones definidas por el usuario.
- Para crear y ejecutar el programa de la función definida por el usuario `UDFsqlsv.java` (función definida por el usuario con sentencias de SQL) en el servidor, realice lo siguiente desde la línea de mandatos:
  1. Compile `UDFsqlsv.java` para crear el archivo `UDFsqlsv.class`, mediante este mandato:
     ```
     javac UDFsqlsv.java
     ```
  2. Copie `UDFsqlsv.class` en el directorio `sqllib\function` (en sistemas operativos Windows) o en el directorio `sqllib/function` (en UNIX).
  3. Compile y ejecute un programa cliente por el que se invoque UDFsqlsv.

     Para acceder a la biblioteca de `UDFsqlsv`, puede utilizar la aplicación `UDFsqlcl.java` de JDBC. El programa cliente contiene la sentencia CREATE FUNCTION de SQL, que permite registrar las funciones definidas por el usuario en la base de datos, y también contiene sentencias de SQL que hacen uso de funciones definidas por el usuario.

Creación de rutinas SQL

Puede utilizar un archivo `makefile` de Java o el archivo de creación `bldsqljs` para crear rutinas SQLJ. Una vez creadas esas rutinas, es necesario catalogarlas.

Los pasos siguientes muestran cómo crear y ejecutar el procedimiento almacenado de ejemplo SpServer de SQLJ. Estos pasos utilizan el archivo de creación bldsqljs (UNIX), o bldsqljs.bat (Windows), que contiene mandatos para crear un applet o aplicación SQLJ.

El archivo de creación utiliza como entrada un máximo de seis parámetros: $1, $2, $3, $4, $5 y $6 en UNIX y %1, %2, %3, %4, %5 y %6 en Windows. El primer parámetro especifica el nombre del programa. El segundo parámetro especifica el ID de usuario correspondiente a la instancia de base de datos; el tercer parámetro especifica la contraseña. El cuarto parámetro especifica el nombre del servidor. El quinto parámetro especifica el número de puerto. Finalmente, el sexto parámetro especifica el nombre de la base de datos. Se pueden utilizar valores por omisión para todos los parámetros, excepto para el primero, el nombre del programa. Consulte el archivo de creación para conocer detalles sobre la utilización de valores de parámetro por omisión.

1. Cree la aplicación del procedimiento almacenado con este mandato:

   ```
   bldsqljs SpServer <ID de usuario> <contraseña> <nombre_servidor>
                                <número_puerto> <nombre_base_datos>
   ```

2. Catalogue el procedimiento almacenado con este mandato:

   ```
   spcat
   ```

   Este script conecta con la base de datos de ejemplo, descataloga mediante SpDrop.db2 las rutinas que se hubieran catalogado previamente, luego las cataloga invocando SpCreate.db2, y finalmente desconecta de la base de datos. También puede ejecutar los scripts SpDrop.db2 y SpCreate.db2 por separado.

3. Detenga y reinicie la base de datos para que se pueda reconocer el nuevo archivo de clase. Si es necesario, configure para lectura la modalidad de archivo del archivo de clase, para que pueda ser leído por el usuario delimitado.
4. Compile y ejecute la aplicación cliente SpClient para acceder a la clase del procedimiento almacenado. Puede crear SpClient mediante el archivo de creación de aplicaciones bldsqlj (UNIX) o bldsqlj.bat (Windows).

Opciones de compilar y enlazar para rutinas Java (SQLJ)

Opciones de rutinas SQLJ para UNIX

El script de creación bldsqljs crea rutinas de SQLJ en los sistemas operativos UNIX. bldsqljs especifica un conjunto de opciones para el traductor y personalizador de SQLJ.

Recomendación: Utilice las mismas opciones para el traductor y personalizador de SQLJ que las utilizadas por bldsqljs cuando cree sus rutinas SQLJ en las plataformas UNIX.

Las opciones incluidas en bldsqljs son:

sqlj Es el traductor SQLJ (también compila el programa).

"${nombreprog}.sqlj"
El archivo fuente de SQLJ. El mandato progname=${1%.sqlj} elimina la extensión si se ha incluido en el nombre de archivo de entrada, por lo que cuando se vuelve a añadir la extensión no está duplicado.

db2sqljcustomize
El personalizador de perfiles SQLJ.

-url Especifica un URL de JDBC para establecer una conexión de base de datos, como jdbc:db2://servername:50000/sample.

-user Especifica un ID de usuario.

-password
Especifica una contraseña.

"${nombreprog}_SJProfile0"
Especifica un perfil serializado para el programa.

Opciones de rutinas SQLJ para Windows

El archivo de proceso por lotes `bldsqljs.bat` crea rutinas SQLJ en los sistemas operativos Windows. `bldsqljs.bat` especifica un conjunto de opciones para el traductor y personalizador de SQLJ.

Recomendación: Utilice las mismas opciones para el traductor y personalizador de SQLJ que las utilizadas por `bldsqljs.bat` cuando cree sus rutinas SQLJ en los sistemas operativos Windows.

Las opciones siguientes del personalizador y el conversor SQLJ se utilizan en el archivo por lotes `bldsqljs.bat` en los sistemas operativos Windows. Estas son las opciones que DB2 recomienda utilizar para crear rutinas SQLJ (procedimientos almacenados y funciones definidas por el usuario).

sqlj El traductor SQLJ (también compila el programa).

%1.sqlj
El archivo fuente de SQLJ.

db2sqljcustomize
El personalizador de perfiles de DB2 para Java.

-url Especifica un URL de JDBC para establecer una conexión de base de datos, como `jdbc:db2://servername:50000/sample`.

-user Especifica un ID de usuario.

-password
Especifica una contraseña.

%1_SJProfile0
Especifica un perfil serializado para el programa.

Despliegue de archivos de clases de rutinas Java en servidores de base de datos DB2

Las implementaciones de rutinas Java deben desplegarse en el sistema de archivos del servidor de bases de datos DB2 de modo que puedan localizarse, cargarse y ejecutarse al invocarse la rutina.

Pueden incluirse una o más implementaciones de rutina Java en un archivo de clase Java individual. Los archivos de clase Java que contienen implementaciones de rutina Java pueden recopilarse en archivos JAR. Los archivos de clases Java que se utilizan para implementar una rutina deben residir en un archivo JAR que habrá instalado en la base de datos DB2.

Requisitos previos:

- Identifique el servidor de datos DB2 en el que desea desplegar la clase de rutina.
- En sistemas operativos UNIX, identifique el ID de usuario del propietario de la instancia de DB2. Si es desconocido, consulte al administrador de bases de datos.

Para desplegar archivos de clase de rutina Java individuales:

- Copie la clase de rutina de Java en el directorio de función de DB2.

  En los sistemas operativos UNIX, el directorio function está definido como: *vía_instalación*/function, donde *vía_instalación* es la vía de instalación del gestor de bases de datos DB2. Por ejemplo, $HOME/sqllib/function, donde $HOME es el directorio inicial del propietario de la instancia.

  En los sistemas operativos Windows, el directorio function está definido como: *vía_perfil_instancia*\function, donde *vía_perfil_instancia* es la vía de acceso especificada en el mandato db2icrt (crear instancia). Para conocer la vía de acceso del perfil de instancia, emita el mandato db2set, del modo siguiente:

  ```
  db2set DB2INSTPROF
  ```

  Por ejemplo, C:\Documents and Settings\All Users\Application Data\IBM\DB2\db2copy1\function

  Si declara que una clase forma parte de un paquete de Java, cree, en el directorio de función, los subdirectorios que correspondan a los nombres de clase completamente calificados y coloque los archivos de clases relacionados en el subdirectorio adecuado. Por ejemplo, si crea una clase llamada ibm.tests.test1 para un sistema operativo Linux, almacene el correspondiente archivo bytecode de Java (llamado test1.class) en $HOME/sqllib/function/ibm/tests, donde $HOME es el directorio inicial del propietario de la instancia.

Para desplegar archivos JAR que contengan archivos de clase de rutina Java:

- Deberá instalar el archivo JAR que contenga los archivos de clase de rutina Java en el sistema de archivos del servidor de bases de datos DB2.

Una vez se hayan desplegado los archivos de clase de rutina Java y que se haya ejecutado la sentencia CREATE para definir la rutina en la base de datos, podrá invocar la rutina.

Despliegue de archivos de clases de rutinas Java en servidores de bases de datos DB2 con clases dependientes

Si los archivos de clases de rutinas Java tienen dependencias en clases que no forman parte de las clases Java o DB2 estándar, repita los pasos identificados en la sección anterior para cada clase de dependencia.

Como alternativa, la base de datos DB2 se puede configurar para que busque en los directorios especificados en la variable de entorno **CLASSPATH** a fin de detectar clases dependientes. En los sistemas operativos Windows, el servidor de bases de datos buscará automáticamente en los directorios especificados en la variable de entorno **CLASSPATH** del sistema. En los sistemas operativos UNIX, el servidor de bases de datos buscará en la variable de entorno **CLASSPATH** del propietario de la instancia si el texto " CLASSPATH " se ha especificado como parte de la variable de entorno **DB2ENVLIST**. Es muy recomendable instalar las clases dependientes, en lugar de depender de la variable de entorno **CLASSPATH**.

Administración de los archivos JAR en el servidor de bases de datos

Para desplegar archivos JAR que contienen archivos de clase de rutina Java, deberá instalar el archivo JAR en el servidor de bases de datos DB2. Esto puede

conseguirse desde IBM Data Server Client utilizando rutinas definidas por el sistema que instalan, sustituyen o eliminan archivos JAR en el servidor de bases de datos de DB2.

Para instalar, sustituir o eliminar un archivo JAR en una instancia de DB2, utilice los procedimientos almacenados que se proporcionan con DB2:

Instalación

```
sqlj.install_jar( url-jar, id-jar )
```

Nota: El ID de autorización de la sentencia del que llama a sqlj.install_jar debe tener al menos uno de los privilegios siguientes:

- Privilegio CREATEIN para el esquema especificado implícita o explícitamente
- Autorización SYSADM o DBADM

Sustitución

```
sqlj.replace_jar( url-jar, id-jar )
```

Eliminación

```
sqlj.remove_jar( id-jar )
```

- *url-jar*: El URL que contiene el archivo JAR que se debe instalar o sustituir. El único esquema de URL que recibe soporte es 'file:'.
- *id-jar*: Un identificador de serie exclusivo, con una longitud de 128 bytes como máximo. Especifica el identificador de JAR incluido en la base de datos que se asocia con el archivo de *url-jar*.

Nota: Cuando se invocan desde aplicaciones, los procedimientos almacenados sqlj.install_jar y sqlj.remove_jar tienen un parámetro adicional. Se trata de un valor de entero que indica el uso del descriptor de despliegue en el archivo JAR especificado. Actualmente, el parámetro de despliegue no está soportado y cualquier invocación que especifique un valor distinto de cero se rechazará.

A continuación, se muestran unos cuantos ejemplos sobre cómo se utilizan los procedimientos almacenados de gestión de archivos JAR citados antes.

Para registrar un JAR ubicado en la vía de acceso `/home/bob/bobsjar.jar` con la instancia de base de datos como MYJAR:

```
CALL sqlj.install_jar( 'file:/home/bob/bobsjar.jar', 'MYJAR' )
```

Los mandatos de SQL posteriores que utilicen el archivo `bobsjar.jar` harán referencia al mismo con el nombre `MYJAR`.

Para sustituir MYJAR por un JAR distinto que contiene algunas clases actualizadas:

```
CALL sqlj.replace_jar( 'file:/home/bob/bobsnewjar.jar', 'MYJAR' )
```

Para eliminar MYJAR de los catálogos de base de datos:

```
CALL sqlj.remove_jar( 'MYJAR' )
```

Nota: En sistemas operativos Windows, DB2 almacena los archivos JAR en la vía de acceso especificada por el valor de registro específico de la instancia *DB2INSTPROF*. Para que los archivos JAR sean exclusivos para una instancia, debe especificar para *DB2INSTPROF* un valor exclusivo que corresponda a dicha instancia.

Actualización de clases de rutina Java

Si desea modificar la lógica de una rutina Java, deberá actualizar el código fuente de la rutina, compile (convierta) el código, y a continuación, actualice la versión de la clase Java o el archivo JAR que se despliega en el servidor de bases de datos DB2. Para asegurarse de que el gestor de bases de datos DB2 utiliza la versión nueva de la rutina Java, debe ejecutar un procedimiento definido por el sistema que cargará la versión nueva de la clase Java en la memoria.

Para actualizar las clases de la rutina Java:

1. Despliegue el archivo de clase Java o el archivo JAR nuevo en el servidor de bases de datos en el servidor de bases de datos DB2.
2. Ejecute el procedimiento definido por el sistema para las rutinas delimitadas:

   ```
   CALL SQLJ.REFRESH_CLASSES()
   ```

   Esta acción forzará al gestor de la base de datos DB2 a cargar la clase nueva en la memoria al producirse la siguiente operación de confirmación o retrotracción.

   Para las rutinas no delimitadas, este paso no funcionará. En las rutinas no delimitadas, deberá detener explícitamente y reiniciar el gestor de bases de datos de DB2 para que se carguen y se utilicen las versiones nuevas de las clases de rutina Java.

Si no realiza los pasos anteriores, después de actualizar las clases de rutinas Java, el gestor de base de datos DB2 seguirá utilizando las versiones anteriores de las clases.

Ejemplos de rutinas Java (JDBC)

Cuando se desarrollan rutinas Java que utilizan la interfaz de programación de aplicaciones, resulta útil hacer referencia a ejemplos para ver el aspecto que puede tener la sentencia CREATE y el código de rutina Java. Los temas siguientes contienen ejemplos de procedimientos y funciones Java:

- Ejemplos de procedimientos Java (JDBC)
- Ejemplos de procedimientos Java (JDBC) con características XML
- Ejemplos de funciones Java (JDBC)

Ejemplo: tipo de datos de matriz en un procedimiento (JDBC) de Java

Ejemplo de una rutina de Java que utiliza el tipo de datos de matriz

En el siguiente ejemplo se muestra el esqueleto de una rutina de Java con un parámetro IN y otro parámetro OUT de un tipo de datos de matriz.

```
CREATE TYPE phonenumbers AS VARCHAR(20) ARRAY[10] %
CREATE PROCEDURE javaproc( IN in1 phonenumbers,
                           OUT out1 phonenumbers)
  LANGUAGE java
  PARAMETER STYLE java
  FENCED THREADSAFE
  EXTERNAL NAME 'myjar:stpclass.javastp' %
```

```
import java.sql.Array;

public static void javaproc(Array input, Array[] output)
{
output[0] = input;
}
```

Ejemplo: Soporte de XML y XQuery en el procedimiento de Java (JDBC)

Una vez comprendidos los conceptos básicos de los procedimientos Java, la programación en Java utilizando la interfaz de programación de aplicaciones (API) de JDBC, y XQuery, podrá empezar a crear y utilizar procedimientos Java que consulten datos XML.

Este ejemplo de procedimiento Java ilustra:

- la sentencia CREATE PROCEDURE para un procedimiento JAVA de estilo de parámetro
- el código fuente para un procedimiento del estilo de parámetro JAVA
- parámetros de entrada y salida de tipo de datos XML
- uso de un parámetro de entrada XML en una consulta
- asignación de una XQuery, un valor XML a un parámetro de salida
- asignación del resultado de una sentencia de SQL, un valor XML a un parámetro de salida

Requisitos previos

Antes de trabajar con este ejemplo de procedimiento Java, puede que sea conveniente leer los temas siguientes:

- Rutinas Rutinas Java
- Rutinas
- Creación de código de rutina Java

Los ejemplos que hay a continuación utilizan una tabla denominada `xmlDataTable` que se define y contiene los datos siguientes:

```
CREATE TABLE xmlDataTable
(
   num INTEGER,
   xdata XML
)@

INSERT INTO xmlDataTable VALUES
   (1, XMLPARSE(DOCUMENT '<doc>
                              <type>car</type>
                              <make>Pontiac</make>
                              <model>Sunfire</model>
                              </doc>' PRESERVE WHITESPACE)),
   (2, XMLPARSE(DOCUMENT '<doc>
                              <type>car</type>
                              <make>Mazda</make>
                              <model>Miata</model>
                              </doc>' PRESERVE WHITESPACE)),
   (3, XMLPARSE(DOCUMENT '<doc>
                              <type>person</type>
                              <name>Mary</name>
                              <town>Vancouver</town>
                              <street>Waterside</street>
                              </doc>' PRESERVE WHITESPACE)),
   (4, XMLPARSE(DOCUMENT '<doc>
                              <type>person</type>
                              <name>Mark</name>
```

```
                                <town>Edmonton</town>
                                <street>Oak</street>
                                </doc>' PRESERVE WHITESPACE)),
        (5, XMLPARSE(DOCUMENT '<doc>
                                <type>animal</type>
                                <name>dog</name>
                                </doc>' PRESERVE WHITESPACE)),
        (6, NULL),
        (7, XMLPARSE(DOCUMENT '<doc>
                                <type>car</type>
                                <make>Ford</make>
                                <model>Taurus</model>
                                </doc>' PRESERVE WHITESPACE)),
        (8, XMLPARSE(DOCUMENT '<doc>
                                <type>person</type>
                                <name>Kim</name>
                                <town>Toronto</town>
                                <street>Elm</street>
                                </doc>' PRESERVE WHITESPACE)),
        (9, XMLPARSE(DOCUMENT '<doc>
                                <type>person</type>
                                <name>Bob</name>
                                <town>Toronto</town>
                                <street>Oak</street>
                                </doc>' PRESERVE WHITESPACE)),
        (10, XMLPARSE(DOCUMENT '<doc>
                                 <type>animal</type>
                                 <name>bird</name>
                                 </doc>' PRESERVE WHITESPACE))@
```

Procedimiento

Utilice el ejemplo siguiente como referencias al crear sus propios procedimientos Java:

- Tabla 42
- Tabla 43 en la página 340

Archivo de código externo Java

En el ejemplo se muestra una implementación de procedimiento Java. El ejemplo se compone de dos partes: la sentencia CREATE PROCEDURE y la implementación de código Java externo del procedimiento desde el que se puede crear la clase Java asociada.

El archivo fuente Java que contiene las implementaciones de procedimientos de los siguientes ejemplos se denomina `stpclass.java` incluido en un archivo JAR denominado `myJAR`. El archivo tiene el formato siguiente:

Tabla 42. Formato del archivo de código externo Java

```
using System;
import java.lang.*;
import java.io.*;
import java.sql.*;
import java.util.*;
import com.ibm.db2.jcc.DB2Xml;

 public class stpclass
 {          ...
    // implementación del procedimiento Java
            ...
 }
```

Las importaciones del archivo de clase Java se indican al principio del archivo. La importación `com.ibm.db2.jcc.DB2Xml` es necesaria si se va a utilizar alguno de los procedimientos del archivo que contenga parámetros o variables del tipo XML.

Es importante tener en cuenta el nombre del archivo de clase y el nombre de JAR que contenga una implementación de procedimiento dada. Estos nombres son importantes, porque la cláusula EXTERNAL de la sentencia CREATE PROCEDURE de cada procedimiento debe especificar esta información a fin de que DB2 pueda localizar la clase en tiempo de ejecución.

Ejemplo 1: Procedimiento del estilo de parámetro JAVA con parámetros XML

Este ejemplo muestra lo siguiente:

- Sentencia CREATE PROCEDURE para un procedimiento de estilo de parámetro JAVA
- Código Java para un procedimiento de estilo de parámetro JAVA con parámetros XML

Este procedimiento toma un parámetro de entrada, `inXML`, inserta una fila que incluya dicho valor en una tabla, consulta datos XML utilizando una sentencia de SQL y una expresión XQuery y establece dos parámetros de salida `outXML1` y `outXML2`.

Tabla 43. Código para crear un procedimiento de estilo de parámetro JAVA con parámetros XML

```
CREATE PROCEDURE  xmlProc1   ( IN inNUM  INTEGER,
                               IN inXML  XML as CLOB (1K),
                               OUT out1XML XML as CLOB (1K),
                               OUT out2XML XML as CLOB (1K)
                                  )
DYNAMIC RESULT SETS 0
DETERMINISTIC
LANGUAGE JAVA
PARAMETER STYLE JAVA
MODIFIES SQL DATA
FENCED
THREADSAFE
DYNAMIC RESULT SETS 0
PROGRAM TYPE SUB
NO DBINFO
EXTERNAL NAME 'myJar:stpclass.xmlProc1'@
```

Tabla 43. Código para crear un procedimiento de estilo de parámetro JAVA con parámetros XML (continuación)

```
//*************************************************************************
//  Stored Procedure: XMLPROC1
//
//  Purpose:   Inserts XML data into XML column; queries and returns XML data
//
//  Parameters:
//
//   IN:      inNum -- the sequence of XML data to be insert in xmldata table
//            inXML -- XML data to be inserted
//   OUT:     out1XML -- XML data to be returned
//            out2XML -- XML data to be returned
//
//*************************************************************************
public void xmlProc1(int inNum,
                     DB2Xml inXML ,
                     DB2Xml[] out1XML,
                     DB2Xml[] out2XML
                    )
throws Exception
{
       Connection con = DriverManager.getConnection( "jdbc:default:connection");

       // Insert data including the XML parameter value into a table
       String query = "INSERT INTO xmlDataTable (num, inXML ) VALUES ( ?, ? )" ;
       String xmlString = inXML.getDB2String() ;

       stmt = con.prepareStatement(query);
       stmt.setInt(1, inNum);
       stmt.setString (2, xmlString );
       stmt.executeUpdate();
       stmt.close();

       // Query and retrieve a single XML value from a table using SQL
       query = "SELECT xdata from xmlDataTable WHERE num = ? " ;

       stmt = con.prepareStatement(query);
       stmt.setInt(1, inNum);
       ResultSet rs = stmt.executeQuery();

       if ( rs.next() )
       {  out1Xml[0] = (DB2Xml) rs.getObject(1);  }

       rs.close();
       stmt.close();

       // Query and retrieve a single XML value from a table using XQuery
       query = "XQUERY for $x in db2-fn:xmlcolumn(\"xmlDataTable.xdata\")/doc
                                where $x/make = \'Mazda\'
                                return <carInfo>{$x/make}{$x/model}</carInfo>";

       stmt = con.createStatement();

       rs = stmt.executeQuery( query );

       if ( rs.next() )
       {  out2Xml[0] = (DB2Xml) rs.getObject(1) ;  }

       rs.close();
       stmt.close();
       con.close();

       return ;
}
```

Capítulo 9. Diseño de rutinas de automatización de OLE

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

Las aplicaciones que exponen las propiedades y los métodos se denominan objetos o servidores de automatización de OLE, y las aplicaciones que acceden a estas propiedades y métodos se denominan controladores de automatización de OLE. Los servidores de automatización de OLE son componentes COM (objetos) que implantan la interfaz IDispatch de OLE. Un controlador de automatización de OLE es un cliente COM que se comunica con el servidor de automatización a través de su interfaz IDispatch. COM es el fundamento de OLE. Para las rutinas de automatización de OLE, DB2 actúa como controlador de automatización de OLE. Mediante este mecanismo, DB2 puede invocar métodos de objetos de automatización de OLE como rutinas externas.

Observe que en todos los temas de automatización de OLE se supone que el usuario está familiarizado con los términos y conceptos de la automatización de OLE. Para tener una visión general de la automatización de OLE, consulte la publicación *Microsoft Corporation: The Component Object Model Specification*, de octubre de 1995. Para conocer detalles sobre la automatización de OLE, consulte la publicación *OLE Automation Programmer's Reference*, Microsoft Press (1996), ISBN 1-55615-851-3.

Creación de rutinas de automatización de OLE

Las rutinas de automatización de OLE se implementan como métodos públicos de los objetos de automatización de OLE. Los objetos de automatización de OLE los tiene que poder crear externamente un controlador de automatización de OLE, en este caso DB2, y soportan un enlace tardío (también denominado enlace basado en IDispatch). Los objetos de automatización de OLE se deben registrar en el registro de Windows con un identificador de clase (CLSID) y, opcionalmente, con un ID programático de OLE (progID) para identificar al objeto de automatización. El progID puede identificar un servidor de automatización de OLE interno del proceso (.DLL) o local (.EXE), o un servidor remoto a través de DCOM (Distributed COM).

Para registrar las rutinas de automatización de OLE:

Después de codificar un objeto de automatización de OLE, tiene que crear los métodos del objeto como rutinas utilizando la sentencia CREATE. Crear rutinas de automatización de OLE es muy similar a registrar rutinas C o C++, pero se deben utilizar las opciones siguientes:

- LANGUAGE OLE
- FENCED NOT THREADSAFE, ya que las rutinas de automatización de OLE se deben ejecutar en modalidad FENCED, pero no se pueden ejecutar como THREADSAFE.

© Copyright IBM Corp. 1993, 2010

El nombre externo consiste en el progID de OLE que identifica el objeto de automatización de OLE y el nombre de método, separados por un ! (signo de exclamación):

```
CREATE FUNCTION bcounter () RETURNS INTEGER
  EXTERNAL NAME 'bert.bcounter!increment'
  LANGUAGE OLE
  FENCED
  NOT THREADSAFE
  SCRATCHPAD
  FINAL CALL
  NOT DETERMINISTIC
  NULL CALL
  PARAMETER STYLE DB2SQL
  NO SQL
  NO EXTERNAL ACTION
  DISALLOW PARALLEL;
```

Los convenios de llamada para las implementaciones de métodos de OLE son idénticos a los utilizados para las rutinas escritas en C o C++. Una implementación del método anterior en el lenguaje BASIC tiene el aspecto siguiente (tenga en cuenta que, en BASIC, los parámetros se definen por omisión como llamadas por referencia):

```
Public Sub increment(output As Long, _
                     indicator As Integer, _
                     sqlstate As String, _
                     fname As String, _
                     fspecname As String, _
                     sqlmsg As String, _
                     scratchpad() As Byte, _
                     calltype As Long)
```

Consideraciones sobre las instancias de objetos de rutinas OLE y el área reutilizable

Los métodos y las UDF de automatización de OLE (métodos de objetos de automatización de OLE) se aplican a instancias de objetos de automatización de OLE. DB2 crea una instancia de un objeto para cada referencia a un método o a una UDF en una sentencia de SQL. Una instancia de un objeto se puede volver a utilizar para invocaciones posteriores al método de referencia del método o de la UDF en una sentencia de SQL, o se puede liberar la instancia después de la invocación al método y crear una nueva instancia para cada invocación posterior al mismo. Se puede especificar el comportamiento adecuado mediante la opción SCRATCHPAD de la sentencia CREATE. Para la cláusula LANGUAGE OLE, la opción SCRATCHPAD tiene una semántica adicional en comparación con C o C++, la cual consiste en que se crea una sola instancia de un objeto y se vuelve a utilizar para la consulta entera, mientras que, si se especifica NO SCRATCHPAD, se puede crear una nueva instancia del objeto cada vez que se invoque un método.

La utilización del área reutilizable permite que un método conserve información sobre el estado en variables de instancias del objeto, a lo largo de las invocaciones a los métodos o funciones. También se incrementa el rendimiento, puesto que sólo se crea una instancia de un objeto una única vez, y luego se vuelve a utilizar para las invocaciones posteriores.

Tipos de datos de SQL soportados en la automatización de OLE

DB2 maneja la conversión de tipos entre tipos de SQL y tipos de automatización de OLE. La tabla siguiente resume los tipos de datos soportados y la forma en que se correlacionan.

Tabla 44. Correlación de tipos de datos de SQL y de la automatización de OLE

Tipo de SQL	Tipo de la automatización de OLE	Descripción del tipo de la automatización de OLE
SMALLINT	short	Entero con signo de 16 bits
INTEGER	long	Entero con signo de 32 bits
REAL	float	Número de coma flotante IEEE de 32 bits
FLOAT o DOUBLE	double	Número de coma flotante IEEE de 64 bits
DATE	DATE	Número de días, fraccionario de coma flotante de 64 bits, desde el 30 de diciembre de 1899
TIME	DATE	
TIMESTAMP	DATE	
CHAR(*n*)	BSTR	Serie de longitud prefijada, descrita en la publicación *OLE Automation Programmer's Reference*.
VARCHAR(*n*)	BSTR	
LONG VARCHAR	BSTR	
CLOB(*n*)	BSTR	
GRAPHIC(*n*)	BSTR	Serie de longitud prefijada, descrita en la publicación *OLE Automation Programmer's Reference*.
VARGRAPHIC(*n*)	BSTR	
LONG GRAPHIC	BSTR	
DBCLOB(*n*)	BSTR	
CHAR(*n*)	SAFEARRAY[unsigned char]	Matriz Byte() de 1 dim de elementos de datos de 8 bits sin signo. (Los tipos SAFEARRAY se describen en la publicación *OLE Automation Programmer's Reference*.)
VARCHAR(*n*)	SAFEARRAY[unsigned char]	
LONG VARCHAR	SAFEARRAY[unsigned char]	
CHAR(*n*) FOR BIT DATA	SAFEARRAY[unsigned char]	
VARCHAR(*n*) FOR BIT DATA	SAFEARRAY[unsigned char]	
LONG VARCHAR FOR BIT DATA	SAFEARRAY[unsigned char]	
BLOB(*n*)	SAFEARRAY[unsigned char]	

Los datos que se pasan entre DB2 y las rutinas de automatización de OLE se pasan como llamadas por referencia. No se soportan los tipos de SQL, por ejemplo BIGINT, DECIMAL o LOCATORS ni los tipos de automatización de OLE tales como Boolean o CURRENCY que no están listados en la tabla. Los datos gráficos y de tipo carácter correlacionados con BSTR se convierten de la página de códigos de la base de datos al esquema UCS-2. (UCS-2 se conoce también como Unicode, página de códigos 13488 de IBM). Al volver, los datos se vuelven a convertir de UCS-2 a la página de códigos de la base de datos. Estas conversiones se producen independientemente de la página de códigos de la base de datos. Si no están instaladas estas tablas de conversión de página de códigos, se recibirá el SQLCODE -332 (SQLSTATE 57017).

Nota: Se sigue ofreciendo soporte para los tipos de datos LONG VARCHAR y LONG VARGRAPHIC, pero éstos se han dejado de utilizar y no se recomiendan ya que puede que se eliminen en un release futuro.

Rutinas de automatización de OLE en BASIC y C++

Puede implementar rutinas de automatización de OLE en cualquier lenguaje. En este apartado se muestra cómo implementar dichas rutinas, utilizando BASIC o C++ como dos lenguajes de ejemplo. La tabla siguiente muestra la correlación de los tipos de automatización de OLE con los tipos de datos en BASIC y C++.

Tabla 45. Correlación de tipos de datos de SQL y OLE con tipos de datos de BASIC y C++

Tipo de SQL	Tipo de la automatización de OLE	Tipo de BASIC	Tipo C++
SMALLINT	short	Integer	short
INTEGER	long	Long	long
REAL	float	Single	float
FLOAT o DOUBLE	double	Double	double
DATE, TIME, TIMESTAMP	DATE	Date	DATE
CHAR(*n*)	BSTR	String	BSTR
CHAR(*n*) FOR BIT DATA	SAFEARRAY[unsigned char]	Byte()	SAFEARRAY
VARCHAR(*n*)	BSTR	String	BSTR
VARCHAR(*n*) FOR BIT DATA	SAFEARRAY[unsigned char]	Byte()	SAFEARRAY
LONG VARCHAR	BSTR	String	BSTR
LONG VARCHAR FOR BIT DATA	SAFEARRAY[unsigned char]	Byte()	SAFEARRAY
BLOB(*n*)	BSTR	String	BSTR
BLOB(*n*) FOR BIT DATA	SAFEARRAY[unsigned char]	Byte()	SAFEARRAY
GRAPHIC(*n*), VARGRAPHIC(*n*), LONG GRAPHIC, DBCLOB(*n*)	BSTR	String	BSTR

Automatización de OLE en BASIC

Para implementar rutinas de automatización de OLE en BASIC, tiene que utilizar los tipos de datos de BASIC que correspondan a los tipos de datos de SQL correlacionados con los tipos de automatización de OLE.

La declaración BASIC de la UDF de automatización de OLE, `bcounter`, tiene el aspecto siguiente:

```
Public Sub increment(output As Long, _
                     indicator As Integer, _
                     sqlstate As String, _
                     fname As String, _
                     fspecname As String, _
                     sqlmsg As String, _
                     scratchpad() As Byte, _
                     calltype As Long)
```

Automatización de OLE en C++

La declaración C++ de la UDF de automatización de OLE, `increment`, es como sigue:

```
STDMETHODIMP Ccounter::increment (long   *output,
                                  short  *indicator,
                                  BSTR   *sqlstate,
                                  BSTR   *fname,
                                  BSTR   *fspecname,
                                  BSTR   *sqlmsg,
                                  SAFEARRAY **scratchpad,
                                  long   *calltype );
```

OLE soporta bibliotecas de tipos que describen las propiedades y los métodos de los objetos de automatización de OLE. Los objetos, las propiedades y los métodos expuestos se describen en el Lenguaje de descripción de objetos (Object Description Language - ODL). La descripción en ODL del método C++ anterior es la siguiente:

```
HRESULT increment ([out]    long  *output,
                   [out]    short *indicator,
                   [out]    BSTR  *sqlstate,
                   [in]     BSTR  *fname,
                   [in]     BSTR  *fspecname,
                   [out]    BSTR  *sqlmsg,
                   [in,out] SAFEARRAY (car. sin signo) *scratchpad,
                   [in]     long *calltype);
```

La descripción en ODL permite especificar si un parámetro es de entrada (in), de salida (out) o de entrada/salida (in,out). Para una rutina de automatización de OLE, los parámetros de entrada a la rutina y los indicadores de entrada se especifican como parámetros [in], y los parámetros de salida de la rutina y los indicadores de salida como parámetros [out]. Para los argumentos de cola de la rutina, `sqlstate` es un parámetro [out], `fname` y `fspecname` son parámetros [in], `scratchpad` es un parámetro [in,out] y `calltype` es un parámetro [in].

La automatización de OLE define el tipo de datos BSTR para manejar series. BSTR se define como puntero a OLECHAR: `typedef OLECHAR *BSTR`. Para asignar y liberar los BSTR, OLE impone la norma de que la rutina llamada libere un BSTR pasado como parámetro por referencia antes de asignar un nuevo valor al parámetro. Se aplica la misma norma para las matrices de bytes unidimensionales recibidas por la rutina llamada como `SAFEARRAY**`. Esta norma significa lo siguiente para DB2 y las rutinas de automatización de OLE:

- Parámetros [in]: DB2 asigna y libera los parámetros [in].
- Parámetros [out]: DB2 pasa un puntero con NULL. La rutina invocada debe asignar el parámetro [out], que DB2 liberará.
- Parámetros [in,out]: DB2 asigna inicialmente los parámetros [in,out]. La rutina invocada puede liberarlos y reasignarlos. Tal como para los parámetros [out], DB2 libera el parámetro final devuelto.

Todos los otros parámetros se pasan como punteros. DB2 asigna y gestiona la memoria referenciada.

La automatización de OLE proporciona un conjunto de funciones de manipulación de datos para tratar los BSTR y SAFEARRAY. Estas funciones se describen en la publicación *OLE Automation Programmer's Reference*.

La rutina en C++ siguiente devuelve los 5 primeros caracteres de un parámetro de entrada CLOB:

```
// UDF DDL: CREATE FUNCTION crunch (CLOB(5k)) RETURNS CHAR(5)

STDMETHODIMP Cobj::crunch (BSTR *in,          // CLOB(5K)
                           BSTR *out,         // CHAR(5)
                           short *indicator1,  // indicador de entrada
```

```
                          short *indicator2,  // indicador de salida
                          BSTR *sqlstate,    // puntero a NULL
                          BSTR *fname,       // puntero a un nombre de función
                          BSTR *fspecname,   // puntero a un nombre específico
                          BSTR *msgtext)     // puntero a NULL
{
   // Asignar BSTR de 5 caracteres
   // y copiar 5 caracteres del parámetro de entrada

   // la salida es un parámetro [out] de tipo BSTR, es decir,
   // es un puntero a NULL y no es necesario liberar la memoria.
   // DB2 liberará el BSTR asignado.

   *out = SysAllocStringLen (*in, 5);
   return NOERROR;
};
```

Un servidor de automatización de OLE se puede implementar como *creatable de un solo uso* o *creatable de varios usos*. Con creatable de un solo uso, cada cliente (es decir, un proceso FENCED de DB2) que conecte mediante `CoGetClassObject` con un objeto de automatización de OLE utilizará su propia instancia de una fábrica de clases, y, de ser necesario, ejecutará una nueva copia del servidor de automatización de OLE. Con creatable de varios usos, muchos clientes se conectan a la misma fábrica de clases. Es decir, a cada creación de una instancia de una fábrica de clases se le suministra una copia del servidor de OLE que ya está en funcionamiento, de haberlas. Si no existen copias del servidor de OLE en funcionamiento, automáticamente se inicia una copia para suministrar el objeto de clase. La elección entre una automatización de OLE de un solo uso o de varios usos corresponde al usuario, en el momento de implementar el servidor de automatización. Para obtener un rendimiento mejor, se recomienda un servidor de un solo uso.

Capítulo 10. Funciones de tabla de OLE DB definidas por el usuario

Microsoft OLE DB es un conjunto de interfaces OLE/COM que proporcionan a las aplicaciones acceso uniforme a datos almacenados en distintas fuentes de información. El componente DBMS de la arquitectura OLE DB define consumidores de OLE DB y proveedores de OLE DB. Un consumidor de OLE DB es cualquier sistema o aplicación que utiliza interfaces de OLE DB; un proveedor de OLE DB es un componente que expone interfaces de OLE DB. Existen dos clases de proveedores de OLE DB: los *proveedores de datos de OLE DB*, que poseen datos y los exponen en formato tabular como un conjunto de filas; y los *proveedores de servicio de OLE DB*, que no poseen sus propios datos, sino que encapsulan algunos servicios produciendo y consumiendo datos a través de las interfaces de OLE DB.

El sistema de bases de datos de DB2 simplifica la creación de aplicaciones OLE DB al permitirle definir funciones de tabla que acceden a una fuente de datos OLE DB. DB2 es un consumidor de OLE DB que puede acceder a cualquier proveedor de servicio o de datos de OLE DB. El usuario puede realizar operaciones que incluyen GROUP BY, JOIN y UNION, sobre fuentes de datos que exponen sus datos a través de interfaces de OLE DB. Por ejemplo, puede definir una función de tabla de OLE DB para devolver una tabla de una base de datos Microsoft Access o de una agenda Microsoft Exchange y luego crear un informe que combine correctamente datos procedentes de esta función de tabla de OLE DB con datos de su base de datos de DB2.

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

Para utilizar las funciones de tabla de OLE DB con DB2, debe instalar OLE DB 2.0 o posteriores, que están disponibles en Microsoft, en la dirección `http://www.microsoft.com`. Si intenta invocar una función de tabla de OLE DB sin instalar primero OLE DB, DB2 emitirá el SQLCODE -465, SQLSTATE 58032, código de razón 35. Para conocer los requisitos del sistema y los proveedores de OLE DB disponibles para las fuentes de datos, consulte la documentación de la fuente de datos. Para ver la especificación de OLE DB, consulte la publicación *Microsoft OLE DB 2.0 Programmer's Reference and Data Access SDK*, Microsoft Press, 1998.

Restricciones para el uso de funciones de tabla de OLE DB: Las funciones de tabla de OLE DB no se pueden conectar a una base de datos de DB2.

© Copyright IBM Corp. 1993, 2010

Creación de una UDF para tablas OLE DB

Para definir una función de tabla OLE DB con una sola sentencia CREATE FUNCTION, debe hacer lo siguiente:

- definir la tabla que devuelve el proveedor de OLE DB
- especificar LANGUAGE OLEDB
- identificar el conjunto de filas de OLE DB y proporcionar una serie de conexión del proveedor de OLE DB en la cláusula EXTERNAL NAME

Las fuentes de datos OLE DB exponen sus datos en forma tabular, llamada *conjunto de filas*. En un conjunto de filas, cada fila tiene un conjunto de columnas. La cláusula RETURNS TABLE sólo devuelve las columnas importantes para el usuario. El enlace de las columnas de la función de tabla con las columnas de un conjunto de filas de una fuente de datos OLE DB se basa en los nombres de columna. Si el proveedor de OLE DB es sensible a las mayúsculas y minúsculas, ponga los nombres de columna entre comillas; por ejemplo, `"UPPERcase"`.
La cláusula EXTERNAL NAME puede tomar cualquiera de las formas siguientes:

```
    'servidor!conjunto-filas'
 o
    '!conjunto-filas!serie-conexión'
```

donde:

servidor
: identifica a un servidor registrado con la sentencia CREATE SERVER

conjunto-filas
: identifica a un conjunto de filas, o una tabla, expuesto por el proveedor de OLE DB; este valor debe estar vacío si la tabla tiene un parámetro de entrada que debe pasar al proveedor de OLE DB a través del texto del mandato.

serie-conexión
: contiene propiedades de inicialización necesarias para conectar con un proveedor de OLE DB. Para ver la sintaxis y la semántica completas de la serie de conexión, consulte el apartado "Data Link API of the OLE DB Core Components" de la publicación *Microsoft OLE DB 2.0 Programmer's Reference and Data Access SDK*, Microsoft Press (1998).

Puede utilizar una *serie de conexión* en la cláusula EXTERNAL NAME de una sentencia CREATE FUNCTION o especificar la opción *CONNECTSTRING* en una sentencia CREATE SERVER.
Por ejemplo, puede definir una función de tabla OLE DB y devolver una tabla desde una base de datos Microsoft Access mediante las sentencias CREATE FUNCTION y SELECT siguientes:

```
CREATE FUNCTION orders ()
     RETURNS TABLE (orderid INTEGER, ...)
     LANGUAGE OLEDB
     EXTERNAL NAME '!orders!Provider=Microsoft.Jet.OLEDB.3.51;
                    Data Source=c:\msdasdk\bin\oledb\nwind.mdb';

   SELECT orderid, DATE(orderdate) AS orderdate,
                   DATE(shippeddate) AS shippeddate
   FROM TABLE(orders()) AS t
   WHERE orderid = 10248;
```

En lugar de colocar la serie de conexión en la cláusula EXTERNAL NAME, puede crear y utilizar un nombre de servidor. Por ejemplo, suponiendo que ha definido el servidor Nwind, puede utilizar la sentencia CREATE FUNCTION siguiente:

```
CREATE FUNCTION orders ()
     RETURNS TABLE (orderid INTEGER, ...)
     LANGUAGE OLEDB
     EXTERNAL NAME 'Nwind!orders';
```

Las funciones de tabla OLE DB también permiten especificar un parámetro de entrada de cualquier tipo de datos de serie de caracteres. Utilice el parámetro de entrada para pasar texto del mandato directamente al proveedor de OLE DB. Si define un parámetro de entrada, no proporcione el nombre de un conjunto de filas en la cláusula EXTERNAL NAME. DB2 pasa el texto del mandato al proveedor de OLE DB para su ejecución, y el proveedor de OLE DB devuelve a DB2 un conjunto de filas. Los nombres de columna y los tipos de datos del conjunto de filas resultante tienen que ser compatibles con la definición RETURNS TABLE de la sentencia CREATE FUNCTION. Puesto que el enlace de los nombres de columna del conjunto de filas se basa en la coincidencia de nombres de columna, se tiene que asegurar de denominar correctamente las columnas.
El ejemplo siguiente registra una función de tabla OLE DB, que recupera información de almacenamiento de una base de datos Microsoft SQL Server 7.0. La serie de conexión se proporciona en la cláusula EXTERNAL NAME. Puesto que la función de tabla tiene un parámetro de entrada que debe pasar al proveedor de OLE DB a través del texto del mandato, el nombre del conjunto de filas no se especifica en la cláusula EXTERNAL NAME. El ejemplo de consulta pasa, en un mandato de SQL, texto que recupera información sobre los tres almacenes superiores de una base de datos SQL Server.

```
CREATE FUNCTION favorites (VARCHAR(600))
  RETURNS TABLE (store_id CHAR (4), name VARCHAR (41), sales INTEGER)
  SPECIFIC favorites
  LANGUAGE OLEDB
  EXTERNAL NAME '!!Provider=SQLOLEDB.1;Persist Security Info=False;
  User ID=sa;Initial Catalog=pubs;Data Source=WALTZ;
  Locale Identifier=1033;Use Procedure for Prepare=1;
  Auto Translate=False;Packet Size=4096;Workstation ID=WALTZ;
  OLE DB Services=CLIENTCURSOR;';

SELECT *
FROM TABLE (favorites (' select top 3 sales.stor_id as store_id,  ' ||
                       '        stores.stor_name as name,         ' ||
                       '        sum(sales. qty) as sales          ' ||
                       ' from sales, stores                       ' ||
                       ' where sales.stor_id = stores.stor_id     ' ||
                       ' group by sales.stor_id, stores.stor_name ' ||
                       ' order by sum(sales.qty) desc')) as f;
```

Nombres de conjunto de filas completamente calificados

Es necesario identificar algunos conjuntos de filas en la cláusula EXTERNAL NAME mediante un *nombre completamente calificado*. Un nombre completamente calificado incorpora cualquiera de los elementos siguientes, o ambos:

- el nombre de catálogo asociado, que requiere la información siguiente:
  - si el proveedor soporta nombres de catálogo

  - dónde se debe situar el nombre de catálogo en el nombre completamente calificado
  - qué separador de nombres de catálogo se debe utilizar
- el nombre de esquema asociado, que requiere la información siguiente:
  - si el proveedor soporta nombres de esquema
  - qué separador de nombres de esquema se debe utilizar

Para obtener información sobre el soporte ofrecido por el proveedor de OLE DB respecto a los nombres de catálogo y de esquema, consulte la documentación de la información literal del proveedor de OLE DB.

Si DBLITERAL_CATALOG_NAME no es NULL en la información literal del proveedor, utilice un nombre de catálogo y el valor de DBLITERAL_CATALOG_SEPARATOR como separador. Para determinar si el nombre de catálogo va al principio o al final del nombre completamente calificado, consulte el valor de DBPROP_CATALOGLOCATION en el conjunto de propiedades DBPROPSET_DATASOURCEINFO del proveedor de OLE DB.

Si DBLITERAL_SCHEMA_NAME no es NULL en la información literal del proveedor, utilice un nombre de esquema y el valor de DBLITERAL_SCHEMA_SEPARATOR como separador.

Si los nombres contienen caracteres especiales o palabras clave coincidentes, encierre los nombres entre las comillas especificadas para el proveedor de OLE DB. Las comillas se definen en la información literal del proveedor de OLE DB como DBLITERAL_QUOTE_PREFIX y DBLITERAL_QUOTE_SUFFIX. Por ejemplo, en el EXTERNAL NAME siguiente, el conjunto de filas especificado incluye el nombre de catálogo *pubs* y el nombre de esquema *dbo* para una fila llamada *authors*, utilizándose la comilla " para encerrar los nombres.

```
EXTERNAL NAME '!"pubs"."dbo"."autores"!Provider=SQLOLEDB.1;...';
```

A fin de obtener más información sobre cómo construir nombres completamente calificados, consulte la publicación *Microsoft OLE DB 2.0 Programmer's Reference and Data Access SDK*, Microsoft Press (1998) así como la documentación correspondiente al proveedor de OLE DB.

Tipos de datos de SQL soportados en OLE DB

La tabla siguiente muestra cómo los tipos de datos de DB2 se correlacionan con los tipos de datos de OLE DB descritos en la publicación *Microsoft OLE DB 2.0 Programmer's Reference and Data Access SDK*, Microsoft Press (1998). Utilice la tabla de correlación para definir las columnas RETURNS TABLE apropiadas en las funciones de tabla OLE DB. Por ejemplo, si se define una función de tabla OLE DB con una columna con tipo de datos INTEGER, DB2 solicita los datos al proveedor de OLE DB como DBTYPE_I4.

Para ver las correlaciones de los tipos de datos fuente del proveedor de OLE DB con los tipos de datos de OLE DB, consulte la documentación del proveedor de OLE DB. Para ver ejemplos de cómo los proveedores de ANSI SQL, Microsoft Access y Microsoft SQL Server pueden correlacionar sus tipos de datos respectivos con los tipos de datos de OLE DB, consulte la publicación *Microsoft OLE DB 2.0 Programmer's Reference and Data Access SDK*, Microsoft Press (1998).

Tabla 46. Correlación de los tipos de datos de DB2 con OLE DB

Tipo de datos de DB2	Tipo de datos de OLE DB
SMALLINT	DBTYPE_I2

Tabla 46. Correlación de los tipos de datos de DB2 con OLE DB (continuación)

Tipo de datos de DB2	Tipo de datos de OLE DB
INTEGER	DBTYPE_I4
BIGINT	DBTYPE_I8
REAL	DBTYPE_R4
FLOAT/DOUBLE	DBTYPE_R8
DEC (p, s)	DBTYPE_NUMERIC (p, s)
DATE	DBTYPE_DBDATE
TIME	DBTYPE_DBTIME
TIMESTAMP	DBTYPE_DBTIMESTAMP
CHAR(N)	DBTYPE_STR
VARCHAR(N)	DBTYPE_STR
LONG VARCHAR	DBTYPE_STR
CLOB(N)	DBTYPE_STR
CHAR(N) FOR BIT DATA	DBTYPE_BYTES
VARCHAR(N) FOR BIT DATA	DBTYPE_BYTES
LONG VARCHAR FOR BIT DATA	DBTYPE_BYTES
BLOB(N)	DBTYPE_BYTES
GRAPHIC(N)	DBTYPE_WSTR
VARGRAPHIC(N)	DBTYPE_WSTR
LONG GRAPHIC	DBTYPE_WSTR
DBCLOB(N)	DBTYPE_WSTR

Nota: Las normas para la conversión de tipos de datos de OLE DB se definen en la publicación *Microsoft OLE DB 2.0 Programmer's Reference and Data Access SDK*, Microsoft Press (1998). Por ejemplo:

- Para recuperar el tipo de datos de OLE DB DBTYPE_CY, los datos se pueden convertir al tipo de datos de OLE DB DBTYPE_NUMERIC(19,4), que está correlacionado con el tipo de datos de DB2 DEC(19,4).
- Para recuperar el tipo de datos de OLE DB DBTYPE_I1, los datos se pueden convertir al tipo de datos de OLE DB DBTYPE_I2, que está correlacionado con el tipo de datos de DB2 SMALLINT.
- Para recuperar el tipo de datos de OLE DB DBTYPE_GUID, los datos se pueden convertir al tipo de datos de OLE DB DBTYPE_BYTES, que se correlaciona con el tipo de datos de DB2 CHAR(12) FOR BIT DATA.

Nota: Se sigue ofreciendo soporte para los tipos de datos LONG VARCHAR y LONG VARGRAPHIC, pero éstos se han dejado de utilizar y no se recomiendan ya que puede que se eliminen en un release futuro.

Capítulo 11. Invocación de rutinas

Una vez desarrollada y creada una rutina en la base de datos emitiendo la sentencia CREATE, si se han otorgado los privilegios adecuados sobre ésta al definidor de la rutina y al invocador de la rutina, se puede invocar la rutina.

Cada tipo de rutina cumple una finalidad distinta y se utiliza de forma distinta. Los requisitos previos para invocar rutinas son comunes, pero la implementación de la invocación difiere para cada una de ellas.

Requisitos previos de la invocación de la rutina

- La rutina se debe haber creado en la base de datos utilizando la sentencia CREATE.
- En el caso de una rutina externa, el archivo de biblioteca o de clases debe estar instalado en la ubicación especificada por la cláusula EXTERNAL de la sentencia CREATE, pues, de lo contrario, se producirá un error (SQLCODE SQL0444, SQLSTATE 42724).
- El invocador de la rutina debe tener el privilegio EXECUTE sobre la rutina. Si el invocador no está autorizado a ejecutar la rutina, se producirá un error (SQLSTATE 42501).

Invocación de procedimientos

Los procedimientos se invocan ejecutando la sentencia CALL con una referencia a un procedimiento.

La sentencia CALL permite la invocación de un procedimiento, el pase de parámetros al procedimiento y la recepción de parámetros devueltos por el procedimiento. Cualquier conjunto de resultados accesible devuelto por un procedimiento se puede procesar una vez que el procedimiento haya realizado su devolución de modo satisfactorio.

Los procedimientos se pueden invocar desde donde esté soportada la sentencia CALL, lo que incluye:

- Aplicaciones cliente
- Rutinas externas (procedimiento, UDF o método)
- Rutinas de SQL (procedimiento, UDF o método)
- Activadores (activadores BEFORE, AFTER o INSTEAD OF)
- Sentencias dinámicas compuestas
- Procesador de línea de mandatos (CLP)

Si elige invocar un procedimiento desde una aplicación cliente o desde una rutina externa, éstas pueden estar escritas en un lenguaje que no sea el del procedimiento. Por ejemplo, una aplicación cliente escrita en C++ puede utilizar la sentencia CALL para invocar un procedimiento escrito en Java. Esto brinda a los programadores una gran flexibilidad para programar en el lenguaje que prefieran y para integrar partes de código escritas en lenguajes distintos.

Además, la aplicación cliente que invoca al procedimiento se puede ejecutar en un sistema operativo distinto de aquél en el que reside el procedimiento. Por ejemplo,

© Copyright IBM Corp. 1993, 2010

una aplicación cliente que se ejecute en un sistema operativo Windows puede utilizar la sentencia CALL para invocar un procedimiento que resida en un servidor de bases de datos Linux.

Según la procedencia de la invocación de un procedimiento, pueden existir diversas consideraciones adicionales.

Invocación de funciones

Las funciones están pensadas para que se haga referencia a ellas dentro de sentencias de SQL.

Se puede hacer referencia a funciones incorporadas, a funciones de agregación con fuente y a funciones escalares definidas por el usuario en cualquier parte donde esté permitida una expresión dentro de una sentencia de SQL. Por ejemplo, dentro de la lista de selección de una consulta o dentro de la cláusula VALUES de una sentencia INSERT. Únicamente se puede hacer referencia a las funciones de tabla en la cláusula FROM. Por ejemplo, en la cláusula FROM de una consulta o de una sentencia de cambio de datos.

Invocación de métodos

Los métodos son parecidos a las funciones escalares, excepto en que éstos se utilizan para proporcionar el comportamiento a los tipos estructurados. La invocación de métodos es como la invocación de funciones escalares definidas por el usuario, con la salvedad de que uno de los parámetros correspondientes al método debe ser el tipo estructurado sobre el que opera el método.

Tareas relacionadas con la invocación de una rutina

Para invocar un tipo determinado de rutina:

- "Llamada a procedimientos desde aplicaciones o rutinas externas" en la página 365
- "Llamada a procedimientos desde activadores o rutinas de SQL" en la página 366
- Consulte el tema "Llamada a un procedimiento desde una aplicación CLI" en *Call Level Interface Guide and Reference, Volume 1*
- "Llamada a procedimientos desde el Procesador de línea de mandatos (CLP)" en la página 368
- "Invocación de funciones escalares o métodos" en la página 377
- "Invocación de funciones de tabla definidas por el usuario" en la página 377

Autorizaciones y enlace de rutinas que contienen SQL

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

Propietario del paquete
El propietario de un paquete en particular que participa en la implementación de una rutina. El propietario del paquete es el usuario que ejecuta el mandato BIND para enlazar un paquete con una base de datos, a menos que se utilice la opción OWNER de precompilación/enlace (BIND) para alterar temporalmente la propiedad del paquete y se establezca en otro usuario. Después de la ejecución del mandato BIND, se otorga el

privilegio EXECUTE WITH GRANT sobre el paquete al propietario del paquete. Una biblioteca de rutinas o un ejecutable puede comprender varios paquetes y, por lo tanto, puede tener asociados varios propietarios de paquetes.

Definidor de la rutina

El ID que emite la sentencia CREATE para registrar una rutina. Generalmente, el definidor de la rutina es un DBA, pero, a menudo, también es el propietario del paquete de la rutina. Cuando se invoca una rutina, durante la carga de los paquetes, la autorización para ejecutarla se comprueba con la autorización del definidor para ejecutar el paquete o paquetes asociados con la rutina (no con la autorización del invocador de la rutina). A fin de que una rutina se invoque satisfactoriamente, el definidor de la misma debe disponer del siguiente privilegio o autorización:

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

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

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

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

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

Invocador de la rutina

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

con el privilegio EXECUTE WITH GRANT OPTION sobre la rutina (esto incluye el definidor de la rutina, a menos que el privilegio se haya revocado explícitamente) o la autorización SYSADM o DBADM emitiendo explícitamente una sentencia GRANT.

Como ejemplo, si un paquete asociado con una aplicación que contiene SQL dinámico se ha enlazado con DYNAMICRULES BIND, su ID de autorización de ejecución será el propietario del paquete, no la persona que invoque el paquete. Asimismo, el propietario del paquete será el enlazador real o el valor de la opción de precompilación/enlace OWNER. En este caso, el invocador de la rutina asume este valor en lugar del ID del usuario que está ejecutando la aplicación.

Nota:

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

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

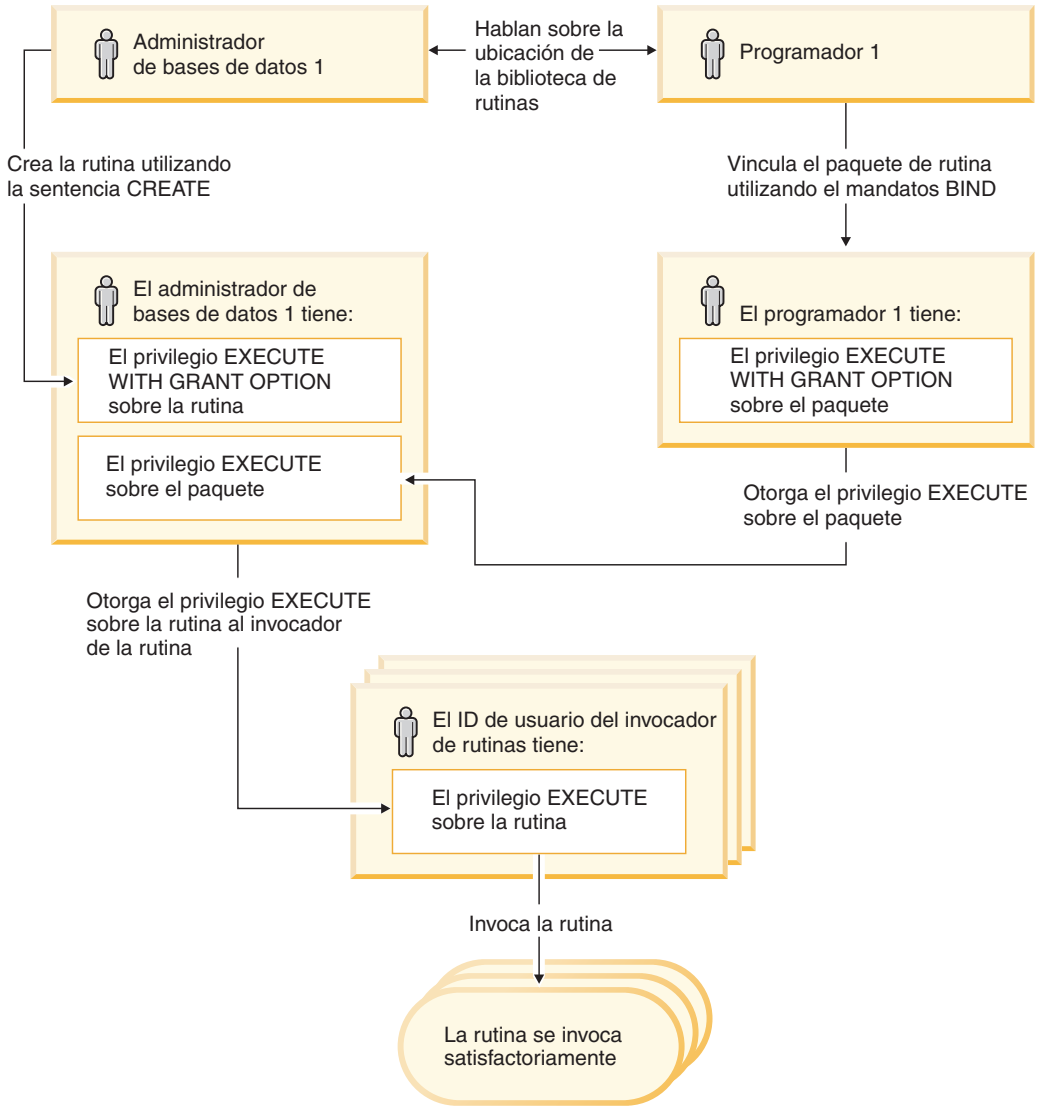


Figura 3. Gestión del privilegio EXECUTE sobre las rutinas

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

respectivo. Este paso sigue al primer paso de esta lista, únicamente para abarcar la posibilidad de una repetición de SQL en la rutina. Si tal repetición no existe en ningún caso en particular, la precompilación/enlace puede preceder a la emisión de la sentencia CREATE para la rutina.

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

Vías de acceso y nombres de rutina

El nombre calificado de un procedimiento almacenado o una UDF es `nombre-esquema.nombre-rutina`. Puede utilizar este nombre calificado en cualquier lugar en que haga referencia a un procedimiento almacenado o a una UDF. Por ejemplo:

```
SANDRA.BOAT_COMPARE    SMITH.FOO    SYSIBM.SUBSTR    SYSFUN.FLOOR
```

No obstante, también puede omitir el `nombre-esquema.`, en cuyo caso, DB2 intentará identificar el procedimiento almacenado o la UDF a la que se hace referencia. Por ejemplo:

```
BOAT_COMPARE    FOO    SUBSTR    FLOOR
```

El nombre calificado de un método es `nombre-esquema.tipo..nombre-método`.

El concepto de *vía de acceso SQL* es vital para la resolución por parte de DB2 de las referencias *no calificadas* que se producen cuando se utiliza el `nombre-esquema`. La vía de acceso SQL es una lista ordenada de nombres de esquema. Proporciona un conjunto de esquemas para resolver las referencias no calificadas a procedimientos almacenados, UDF y tipos. En los casos en que una referencia coincide con un procedimiento almacenado, tipo o UDF en más de un esquema de la vía de acceso, se utiliza el orden de los esquemas de la vía de acceso para resolver esta coincidencia. La vía de acceso SQL se establece por medio de la función FUNCPATH en los mandatos de precompilación y enlace del SQL estático. En el SQL dinámico, se establece mediante la sentencia SET PATH. La vía de acceso SQL tiene el valor por omisión siguiente:

```
"SYSIBM","SYSFUN","SYSPROC", "ID"
```

Esto es aplicable al SQL tanto estático como dinámico, siendo que *ID* representa el ID de autorización de la sentencia actual.

Los nombres de rutina se pueden *sobrecargar*, lo que significa que múltiples rutinas, incluso del mismo esquema, pueden tener el mismo nombre. Múltiples funciones o métodos con el mismo nombre pueden tener el mismo número de parámetros, siempre y cuando los tipos de datos sean distintos. Esto no se aplica a los

procedimientos almacenados, ya que varios procedimientos almacenados con el mismo nombre deben tener un número distinto de parámetros. Las instancias de distintos tipos de rutinas no se sobrecargan entre sí, a excepción de los métodos, los cuales pueden sobrecargar las funciones. Para que un método sobrecargue una función, el método se debe registrar utilizando la cláusula WITH FUNCTION ACCESS.

Una función, un procedimiento almacenado y un método pueden tener *signaturas* idénticas y estar en el mismo esquema sin sobrecargarse entre sí. En el contexto de las rutinas, las signaturas son el nombre de rutina calificado que se concatena con los tipos de datos definidos de todos los parámetros en el orden en que se han definido.

Los métodos se invocan sobre instancias de su tipo estructurado asociado. Cuando se crea un subtipo, entre los atributos que éste hereda se encuentran los métodos definidos para el supertipo. Por ello, los métodos de un supertipo también se pueden ejecutar sobre las instancias de sus subtipos. Al definir un subtipo, se puede *alterar temporalmente* el método del supertipo. Alterar temporalmente un método significa volverlo a implementar específicamente para un subtipo determinado. Esto facilita el despacho dinámico de métodos (también conocido como polimorfismo), en que una aplicación ejecutará el método más específico según el tipo de instancia del tipo estructurado (por ejemplo, en qué lugar de la jerarquía de tipos estructurados está situado).

Cada tipo de rutina tiene su propio algoritmo de selección que tiene en cuenta los hechos de sobrecarga (y, en el caso de los métodos, la alteración temporal) y la vía de acceso SQL a fin de elegir la coincidencia más adecuada para cada referencia de rutina.

Invocaciones de rutinas anidadas

En el contexto de las rutinas, el *anidamiento* hace referencia a una situación en que una rutina invoca a otra. Es decir, que el SQL emitido por una rutina puede hacer referencia a otra rutina, la cual puede emitir SQL que haga de nuevo referencia a otra rutina, y así sucesivamente. Si la serie de rutinas referida contiene una rutina referida anteriormente, se considera que es una situación de anidamiento *repetitivo*.

Puede utilizar el anidamiento y la repetición en las rutinas de DB2 con las restricciones siguientes:

64 niveles de anidamiento
Puede anidar invocaciones a rutinas hasta un máximo de 64 niveles de profundidad. Suponga que la rutina A llama a la rutina B y ésta llama a la rutina C. En este ejemplo, la ejecución de la rutina C está en el nivel de anidamiento 3. Son posibles 61 niveles de anidamiento más.

Otras restricciones
Una rutina no puede llamar a una rutina de destino que esté catalogada con un nivel de acceso a datos SQL superior. Por ejemplo, una UDF creada con la cláusula CONTAINS SQL puede llamar a procedimientos almacenados creados con la cláusula CONTAINS SQL o la cláusula NO SQL. Sin embargo, esta rutina no podrá llamar a procedimientos almacenados creados con la cláusula READS SQL DATA o la cláusula MODIFIES SQL DATA (SQLCODE -577, SQLSTATE 38002). Es así porque el nivel de SQL del invocador no permite que tenga lugar ninguna operación de lectura o modificación (esto lo hereda la rutina invocada).

Otra limitación al anidar rutinas es que el acceso a tablas está restringido, a fin de evitar operaciones conflictivas de lectura y grabación entre las rutinas.

Invocación de rutinas de 32 bits en un servidor de bases de datos de 64 bits

En instancias de DB2 de 64 bits, es posible invocar rutinas C y COBOL que hagan referencia a bibliotecas de rutinas externas de 32 bits; sin embargo, debe especificarse que estas rutinas se ejecuten como fenced y not threadsafe. Esto se consigue incluyendo las cláusulas FENCED y NOT THREADSAFE en la sentencia CREATE de la rutina al crear una rutina nueva. En el caso de rutinas que ya se hayan creado en una instancia de 64 bits, pueden utilizarse las sentencias ALTER FUNCTION o ALTER PROCEDURE para modificar la definición de la rutina. La primera vez que se invoca una rutina de 32 bits en un entorno de 64 bits, se producirá una disminución del rendimiento. Las invocaciones posteriores del procedimiento almacenado de 32 bits funcionarán igual que una rutina de 64 bits equivalente. Se desaconseja el uso de rutinas de 32 bits en instancias de DB2 de 64 bits.

Para invocar satisfactoriamente procedimientos Java en una instancia de DB2 de 64 bits en un servidor de bases de datos de 64 bits, se necesita una máquina virtual de Java (JVM) de 64 bits. Las JVM de 32 bits no están soportadas para ejecutar rutinas en instancias de DB2 de 64 bits. Puesto que las clases de Java son independientes de la plataforma, una clase de Java compilada con un kit de desarrollo de software de 32 bits puede ejecutarse correctamente con una JVM de 64 bits. El rendimiento de la rutina no se ve afectado por ello.

Para invocar rutinas de 32 bits existentes en un servidor de 64 bits:

1. Copie la clase o la biblioteca de la rutina en el directorio de rutinas de base de datos:
   - UNIX: sqllib/function
   - Windows: sqllib\function
2. Registre el procedimiento almacenado con la sentencia CREATE PROCEDURE.
3. Invoque al procedimiento almacenado con la sentencia CALL.

Consideraciones sobre la página de códigos de las rutinas

Los datos de tipo carácter se pasan a las rutinas externas en la página de códigos implicada en la opción PARAMETER CCSID que se ha utilizado al crear la rutina. De forma similar, la base de datos supone que la serie de caracteres de salida de la rutina utiliza la página de códigos implicada en la opción PARAMETER CCSID.

Cuando un programa cliente (que utiliza, por ejemplo, la página de códigos C) accede a una sección con una página de códigos diferente (por ejemplo, la página de códigos S) que invoca a una rutina que utiliza otra página de códigos (por ejemplo, la página de códigos R), se producen los sucesos siguientes:

1. Cuando se invoca una sentencia de SQL, los datos de tipo carácter de entrada se convierten, de la página de códigos de la aplicación cliente (C) a la página de códigos asociada con la sección (S). La conversión no tiene lugar para los BLOB ni los datos que se utilizarán como FOR BIT DATA.

2. Si la página de códigos de la rutina no es la de la sección, entonces, antes de que se invoque la rutina, los datos de tipo carácter de entrada (excepto BLOB y FOR BIT DATA) se convierten a la página de códigos de la rutina (R).

   Es muy recomendable que precompile, compile y enlace la rutina del servidor utilizando la página de códigos bajo la que se invocará la rutina (R). Esto puede no ser posible en todos los casos. Por ejemplo, se puede crear una base de datos Unicode en un entorno Windows. No obstante, si el entorno Windows no tiene la página de códigos Unicode, se tendrá que precompilar, compilar y enlazar la aplicación que cree la rutina en una página de códigos de Windows. La rutina funcionará si la aplicación no tiene ningún carácter delimitador especial que el precompilador no comprenda.
3. Cuando la rutina finaliza, el gestor de bases de datos convierte todos los datos de tipo carácter de salida, de la página de códigos de la rutina (R) a la página de códigos de la sección (S), si es necesario. Si la rutina ha emitido un error durante su ejecución, el SQLSTATE y el mensaje de diagnóstico de la rutina también se convertirán, de la página de códigos de la rutina a la página de códigos de la sección. No tiene lugar ninguna conversión para las series de caracteres BLOB y FOR BIT DATA.
4. Cuando la sentencia finaliza, los datos de tipo carácter de salida se convierten, de la página de códigos de la sección (S) a la página de códigos de la aplicación cliente (C) otra vez. La conversión no tiene lugar para los BLOB ni para los datos utilizados como FOR BIT DATA.

Utilizando la opción DBINFO en las sentencias CREATE FUNCTION, CREATE PROCEDURE y CREATE TYPE, se pasa a la rutina su página de códigos. Mediante el uso de esta información, una rutina sensible a la página de códigos se puede escribir de forma que funcione en muchas páginas de códigos distintas.

Referencias a procedimientos

Los procedimientos almacenados se invocan desde la sentencia CALL, en la que se les hace referencia con un nombre calificado (nombre de esquema y nombre del procedimiento almacenado), seguido de una lista de argumentos encerrados entre paréntesis. Un procedimiento almacenado también se puede invocar sin el nombre de esquema, lo que da como resultado una elección entre posibles procedimientos almacenados de distintos esquemas con el mismo número de parámetros.

Cada parámetro pasado al procedimiento almacenado puede estar compuesto por una variable del lenguaje principal, un marcador de parámetro, una expresión o un NULL. A continuación, se indican las restricciones correspondientes a los parámetros de procedimiento almacenado:

- Los parámetros OUT e INOUT deben ser variables del lenguaje principal.
- No se pueden pasar NULL a los procedimientos almacenados Java a menos que el tipo de datos de SQL esté correlacionado con un tipo de clase de Java.
- No se pueden pasar NULL a los procedimientos almacenados PARAMETER STYLE GENERAL.

La posición de los argumentos es importante y se debe ajustar a la definición del procedimiento almacenado para que la semántica sea correcta. Tanto la posición de los argumentos como la definición del procedimiento almacenado se deben ajustar al propio cuerpo del procedimiento almacenado. DB2 no intenta cambiar de sitio los argumentos para que coincidan mejor con una definición de procedimiento almacenado y DB2 no comprende la semántica de los parámetros de procedimiento almacenado individuales.

Selección de procedimiento

Dada una invocación de procedimiento almacenado, el gestor de bases de datos debe decidir a cuál de los posibles procedimientos almacenados con el mismo nombre debe llamar. La resolución del procedimiento almacenado se realiza siguiendo los pasos explicados a continuación.

1. Se buscan todos los procedimientos almacenados del catálogo (SYSCAT.ROUTINES) que cumplan las condiciones siguientes:
   - Para las invocaciones en que se ha especificado el nombre de esquema (es decir, las referencias calificadas), el nombre de esquema y el nombre del procedimiento almacenado coinciden con el nombre de invocación.
   - Para las invocaciones en que el nombre de esquema no se ha especificado (es decir, las referencias no calificadas), el nombre del procedimiento almacenado coincide con el nombre de invocación y tiene un nombre de esquema que coincide con uno de los esquemas de la vía de acceso SQL.
   - El número de parámetros definidos coincide con la invocación.
   - El invocador tiene el privilegio EXECUTE sobre el procedimiento almacenado.
2. Se elige el procedimiento almacenado cuyo esquema vaya primero en la vía de acceso SQL.

Si no quedan procedimientos almacenados candidatos después del primer paso, se devolverá un error (SQLSTATE 42884).

Llamada a procedimientos

Una vez que las actividades necesarias para crear un procedimiento (también denominado procedimiento almacenado) se han completado, un procedimiento se puede invocar utilizando la sentencia CALL. La sentencia CALL es una sentencia de SQL que permite la invocación de un procedimiento, el pase de parámetros al procedimiento y la recepción de parámetros devueltos por el procedimiento. Cualquier conjunto de resultados accesible devuelto por un procedimiento se puede procesar una vez que el procedimiento haya realizado su devolución de modo satisfactorio. Los procedimientos se pueden invocar desde donde esté soportada la sentencia CALL, lo que incluye:

- una aplicación cliente de SQL incorporado
- una rutina externa (procedimiento, UDF o método)
- una rutina de SQL (procedimiento, UDF o método)
- un activador de SQL (BEFORE TRIGGER, AFTER TRIGGER o INSTEAD OF TRIGGER)
- una sentencia compuesta dinámica de SQL
- desde el procesador de línea de mandatos (CLP)

Si elige invocar un procedimiento desde una aplicación cliente o desde una rutina externa, éstas pueden estar escritas en un lenguaje que no sea el del procedimiento. Por ejemplo, una aplicación cliente escrita en C++ puede utilizar la sentencia CALL para invocar un procedimiento escrito en Java. Esto brinda a los programadores una gran flexibilidad para programar en el lenguaje que prefieran y para integrar partes de código escritas en lenguajes distintos.

Además, la aplicación cliente que invoca el procedimiento se puede ejecutar en una plataforma distinta de aquélla en la que reside el procedimiento. Por ejemplo, una

aplicación cliente que se ejecute en una plataforma Windows puede utilizar la sentencia CALL para invocar un procedimiento que resida en un servidor de bases de datos Linux.

Para saber cómo puede llamar a un procedimiento desde una interfaz determinada, consulte:

- "Llamada a procedimientos desde activadores o rutinas de SQL" en la página 366
- "Llamada a procedimientos desde aplicaciones o rutinas externas"
- "Llamada a procedimientos almacenados desde aplicaciones CLI" en *Call Level Interface Guide and Reference, Volume 1*
- "Llamada a procedimientos desde el Procesador de línea de mandatos (CLP)" en la página 368

Llamada a procedimientos desde aplicaciones o rutinas externas

La invocación de un procedimiento (también denominado procedimiento almacenado) que encapsula lógica de una aplicación cliente o de una aplicación asociada con una rutina externa, se realiza fácilmente con un simple trabajo de configuración en la aplicación y mediante la sentencia CALL.

El procedimiento se debe haber creado en la base de datos ejecutando la sentencia CREATE PROCEDURE.

Para los procedimientos externos, el archivo de biblioteca o de clases debe existir en la ubicación especificada por la cláusula EXTERNAL de la sentencia CREATE PROCEDURE.

El invocador del procedimiento debe tener los privilegios necesarios para ejecutar la sentencia CALL. El invocador del procedimiento, en este caso, es el ID de usuario que ejecuta la aplicación; no obstante, se aplican reglas especiales si se utiliza la opción de enlace DYNAMICRULES para la aplicación.

Se deben incluir ciertos elementos en la aplicación si desea que ésta invoque a un procedimiento. Al escribir la aplicación, debe realizar lo siguiente:

1. Declare, asigne e inicialice el almacenamiento para las estructuras de datos opcionales y las variables del lenguaje principal o marcadores de parámetros que necesita la sentencia CALL. Para ello:
   - Asigne una variable del lenguaje principal o marcador de parámetro que se utilice para cada parámetro del procedimiento.
   - Inicialice las variables del lenguaje principal o los marcadores de parámetros correspondientes a los parámetros IN o INOUT.
2. Establezca una conexión de base de datos. Esto se consigue ejecutando una sentencia CONNECT TO en el lenguaje SQL incorporado o bien codificando una conexión de base de datos implícita.
3. Codifique la invocación del procedimiento. Después de la conexión de base de datos, puede codificar la invocación del procedimiento. Para ello, ejecute la sentencia CALL en el lenguaje SQL. Asegúrese de especificar una variable del lenguaje principal, constante o marcador de parámetro para cada parámetro IN, INOUT y OUT que espere el procedimiento.
4. Añada código para procesar los parámetros OUT e INOUT y los conjuntos de resultados. Este código debe ser posterior a la ejecución de la sentencia CALL.

5. Codifique una operación de base de datos COMMIT o ROLLBACK. Después de la sentencia CALL y de la evaluación de los valores de parámetro de salida o datos devueltos por el procedimiento, puede ser conveniente que la aplicación confirme o retrotraiga la transacción. Para ello, se incluye una sentencia COMMIT o ROLLBACK. Un procedimiento puede incluir una sentencia COMMIT o ROLLBACK, pero es una práctica recomendable que la gestión de transacciones se realice dentro de la aplicación cliente.

   Nota: Los procedimientos invocados desde una aplicación que haya establecido una conexión de tipo 2 con la base de datos no pueden emitir sentencias COMMIT ni ROLLBACK.
6. Desconéctese de la base de datos.
7. Prepare, compile, enlace y vincule la aplicación. Si la aplicación corresponde a una rutina externa, emita la sentencia CREATE a fin de crear la rutina y ubique la biblioteca de código externo en la vía de acceso de función adecuada para el sistema operativo de modo que el gestor de bases de datos la pueda encontrar.
8. Ejecute la aplicación o invoque la rutina externa. Se invocará la sentencia CALL que ha incorporado en la aplicación.

Nota: Puede codificar sentencias de SQL y lógica de rutina en cualquier momento entre los pasos 2 y 5.

Llamada a procedimientos desde activadores o rutinas de SQL

Llamar a un procedimiento desde una rutina de SQL, un activador o una sentencia dinámica compuesta es, básicamente, lo mismo. Se utilizan pasos iguales para implementar esta llamada. Este tema explica los pasos utilizando un escenario de activador. Se indica cualquier requisito previo o paso que difiera al llamar a un procedimiento desde una rutina o sentencia dinámica compuesta.

Requisitos previos

- El procedimiento se debe haber creado en la base de datos ejecutando la sentencia CREATE PROCEDURE.
- Para los procedimientos externos, los archivos de biblioteca o clases se deben encontrar en la ubicación especificada por la cláusula EXTERNAL de la sentencia CREATE PROCEDURE.
- El creador de un activador que contenga una sentencia CALL debe tener el privilegio de ejecutar la sentencia CALL. Durante la ejecución, cuando se activa un activador, es en la autorización del creador del activador en la que se comprueba el privilegio para ejecutar la sentencia CALL. Un usuario que ejecute una sentencia dinámica compuesta que contenga una sentencia CALL debe tener el privilegio de ejecutar la sentencia CALL para ese procedimiento.
- Para invocar un activador, un usuario debe tener el privilegio de ejecutar la sentencia de cambio de datos asociada con el suceso activador. De forma similar, para invocar satisfactoriamente una rutina de SQL o sentencia dinámica compuesta, un usuario debe tener el privilegio EXECUTE sobre la rutina.

Cuando se invoca un procedimiento desde un activador de SQL, una rutina de SQL o una sentencia dinámica compuesta, se aplican las restricciones siguientes:

- En los entornos de bases de datos particionadas, no es posible invocar procedimientos desde activadores o UDF de SQL.
- En las máquinas de multiprocesador simétrico (SMP), las llamadas de procedimiento desde activadores se ejecutan en un solo procesador.

- Un procedimiento que se ha de llamar desde un activador no debe contener una sentencia COMMIT ni una sentencia ROLLBACK que intente retrotraer la unidad de trabajo. La sentencia ROLLBACK TO SAVEPOINT está soportada dentro del procedimiento; no obstante, el punto de salvaguarda especificado debe estar en el procedimiento.
- Una retrotracción de una sentencia CALL desde un activador no retrotraerá ninguna acción externa afectada por los procedimientos, como la grabación en el sistema de archivos.
- El procedimiento no debe modificar ninguna tabla federada. Esto significa que el procedimiento no debe contener una actualización (UPDATE) buscada de un apodo, una supresión (DELETE) buscada desde un apodo ni una inserción (INSERT) en un apodo.
- Los conjuntos de resultados especificados para el procedimiento no serán accesibles.

Los activadores BEFORE no se pueden crear si contienen una sentencia CALL que hace referencia a un procedimiento creado con un nivel de acceso de MODIFIES SQL DATA. La ejecución de una sentencia CREATE TRIGGER para un activador de ese tipo fallará con el error (SQLSTATE 42987). Si desea más información sobre los niveles de acceso de SQL en las rutinas, consulte:

- "Niveles de acceso de SQL en rutinas" en la página 43
- "Sentencias de SQL que pueden ejecutarse en rutinas" en la página 38

Este apartado sobre el procedimiento explica cómo crear e invocar un activador que contenga una sentencia CALL. El SQL necesario para llamar a un procedimiento desde un activador es el mismo que se necesita para llamar a un procedimiento desde una rutina de SQL o sentencia dinámica compuesta.

1. Escriba una sentencia CREATE TRIGGER básica especificando los atributos deseados del activador. Consulte la forma de crear la sentencia CREATE TRIGGER.
2. En la parte de la acción activadora del activador, puede declarar variables de SQL para cualquier parámetro IN, INOUT u OUT especificado por el procedimiento. Consulte la forma de crear la sentencia DECLARE. Para conocer cómo se inicializan o establecen estas variables, consulte la sentencia de asignación. Las variables de transición de activador también se pueden utilizar como parámetros para un procedimiento.
3. En la parte de la acción activadora del activador, añada una sentencia CALL para el procedimiento. Especifique un valor o expresión para cada uno de los parámetros IN, INOUT y OUT del procedimiento.
4. Para los procedimientos de SQL, puede capturar opcionalmente el estado de retorno del procedimiento utilizando la sentencia GET DIAGNOSTICS. Con este fin, tendrá que emplear una variable de tipo entero que incluya el estado de retorno. Inmediatamente después de la sentencia CALL, añada simplemente una sentencia GET DIAGNOSTICS que asigne RETURN_STATUS a la variable de estado de retorno del activador local.
5. Una vez que haya terminado de escribir la sentencia CREATE TRIGGER, ahora puede ejecutarla de forma estática (desde una aplicación) o dinámica (desde el CLP o desde el Centro de control) para crear formalmente el activador en la base de datos.
6. Invoque el activador. Para ello, realice la ejecución sobre la sentencia de cambio de datos adecuada que corresponda al suceso activador.

7. Cuando la sentencia de cambio de datos se ejecuta sobre la tabla, se desencadenan los activadores adecuados definidos para dicha tabla. Cuando se ejecuta la acción activadora, se ejecutan las sentencias de SQL que contiene ésta, incluida la sentencia CALL.

Errores de ejecución

Si el procedimiento intenta leer o grabar en una tabla en la que el activador también lee o graba, se puede emitir un error en el caso de que se detecte un conflicto de lectura o grabación. El conjunto de tablas que modifica el activador, incluida la tabla para la cual se ha definido el activador, se debe excluir de las tablas modificadas por el procedimiento.

Ejemplo: Llamada a un procedimiento de SQL desde un activador

Este ejemplo ilustra cómo puede intercalar una sentencia CALL para invocar un procedimiento dentro de un activador y cómo se captura el estado de retorno de la llamada de procedimiento utilizando la sentencia GET DIAGNOSTICS. El siguiente código SQL crea las tablas necesarias, un procedimiento de lenguaje SQL PL y un activador AFTER.

```
CREATE TABLE T1 (c1 INT, c2 CHAR(2))@
CREATE TABLE T2 (c1 INT, c2 CHAR(2))@

CREATE PROCEDURE proc(IN val INT, IN name CHAR(2))
LANGUAGE SQL
DYNAMIC RESULTSETS 0
MODIFIES SQL DATA
BEGIN
  DECLARE rc INT DEFAULT 0;
  INSERT INTO TABLE T2 VALUES (val, name);
  GET DIAGNOSTICS rc = ROW_COUNT;
  IF ( rc > 0 ) THEN
      RETURN 0;
  ELSE
      RETURN -200;
  END IF;
END@

CREATE TRIGGER trig1 AFTER UPDATE ON t1
REFERENCING NEW AS n
FOR EACH ROW
WHEN (n.c1 > 100);
BEGIN ATOMIC
   DECLARE rs INTEGER DEFAULT 0;
   CALL proc(n.c1, n.c2);
   GET DIAGNOSTICS rs = RETURN_STATUS;
   VALUES(CASE WHEN rc < 0 THEN RAISE_ERROR('70001', 'PROC CALL failed'));
END@
```

La emisión de la sentencia de SQL siguiente causará que el activador se desencadene y que se invoque el procedimiento.

```
UPDATE T1 SET c1 = c1+1 WHERE c2 = 'CA'@
```

Llamada a procedimientos desde el Procesador de línea de mandatos (CLP)

Puede llamar a procedimientos almacenados utilizando la sentencia `CALL` desde la interfaz del procesador de línea de mandatos de DB2. El procedimiento almacenado al que se llame debe estar definido en las tablas de catálogo del sistema DB2.

Para llamar al procedimiento almacenado, primero conéctese a la base de datos:

```
db2 connect to sample user IDusuario using contraseña
```

donde *IDusuario* y *contraseña* son el ID de usuario y la contraseña de la instancia donde reside la base de datos `sample`.
Para utilizar la sentencia `CALL`, entre el nombre del procedimiento almacenado más cualquier valor de parámetro IN o INOUT, así como el signo '?' en calidad de contenedor de cada valor de parámetro OUT.
Los parámetros de un procedimiento almacenado se proporcionan en la sentencia `CREATE PROCEDURE` para dicho procedimiento en el archivo fuente del programa.

Ejemplos de procedimientos de SQL

Ejemplo 1.

En el archivo `whiles.db2`, la sentencia `CREATE PROCEDURE` para la signatura del procedimiento `DEPT_MEDIAN` es la siguiente:

```
CREATE PROCEDURE DEPT_MEDIAN
(IN deptNumber SMALLINT, OUT medianSalary DOUBLE)
```

Para invocar este procedimiento, utilice la sentencia CALL en la que debe especificar el nombre de procedimiento y los argumentos de parámetro apropiados, que en este caso, son el valor del parámetro IN y un signo de interrogación, '?', para el valor del parámetro OUT. La sentencia SELECT del procedimiento utiliza el valor `deptNumber` de la columna DEPT de la tabla STAFF, por ello, a fin de obtener una salida significativa, el parámetro IN tiene que ser un valor correcto de la columna DEPT; por ejemplo, el valor "51":

```
db2 call dept_median (51, ?)
```

Nota: En las plataformas UNIX, los paréntesis tienen un significado especial para el shell de mandatos, por lo que deben ir precedidos del carácter "\" o se deben encerrar entre comillas, del modo siguiente:

```
db2 "call dept_median (51, ?)"
```

No utilice comillas si se ha de utilizar la modalidad interactiva del procesador de línea de mandatos.
Después de ejecutar el mandato anterior, tendría que recibir un resultado como el siguiente:

```
Value of output parameters
--------------------------
Parameter Name  : MEDIANSALARY
Parameter Value : +1.76545000000000E+004

Return Status = 0
```

Ejemplo 2.

En este ejemplo se muestra la forma en que llamar a un procedimiento con parámetros de tipo matriz El tipo phonenumbers se define como:

```
CREATE TYPE phonenumbers AS VARCHAR(12) ARRAY[1000]
```

El procedimiento find_customers, que se define más adelante, tiene un parámetro IN y un parámetro OUT, los dos del tipo phonenumbers. El procedimiento busca números en numbers_in que empiecen por un area_code determinado, e informa sobre los mismos en numbers_out.

```
CREATE PROCEDURE find_customers(
IN numbers_in phonenumbers,
IN area_code CHAR(3),
OUT numbers_out phonenumbers)
BEGIN
```

```
DECLARE i, j, max INTEGER;

SET i = 1;
SET j = 1;
SET numbers_out = NULL;
SET max = CARDINALITY(numbers_in);

WHILE i <= max DO
IF substr(numbers_in[i], 1, 3) = area_code THEN
SET numbers_out[j] = numbers_in[i];
SET j = j + 1;
END IF;
SET i = i + 1;
END WHILE;
END
```

Utilizando la sentencia CALL que se muestra a continuación, se invoca al procedimiento:

```
db2 CALL find_customers(ARRAY['416-305-3745',
                              '905-414-4565',
                              '416-305-3746'],
                              '416',
                               ?)
```

Tal como se muestra en la sentencia CALL, cuando un procedimiento tiene un parámetro de entrada del tipo de datos de matriz, el argumento de entrada se puede especificar con un constructor de matriz con una lista de valores literales.

Después de ejecutar el mandato, debería obtener un resultado parecido a este:

```
Value of output parameters
  --------------------------
  Parameter Name  : OUT_PHONENUMBERS
  Parameter Value : ['416-305-3745',
                     '416-305-3746']

  Return Status = 0
```

Ejemplo de procedimiento almacenado C

Con el Procesador de línea de mandatos, también puede llamar a procedimientos almacenados creados a partir de los lenguajes principales soportados. En el directorio `samples/c` de UNIX y en el directorio `samples\c` de Windows, DB2 proporciona archivos de creación de procedimientos almacenados. La biblioteca compartida `spserver` contiene varios procedimientos almacenados que se pueden crear a partir del archivo fuente, `spserver.sqc`. El archivo `spcreate.db2` cataloga los procedimientos almacenados.

En el archivo `spcreate.db2`, la sentencia `CREATE PROCEDURE` para el procedimiento `MAIN_EXAMPLE` empieza así:

```
CREATE PROCEDURE MAIN_EXAMPLE (IN job CHAR(8),
                               OUT salary DOUBLE,
                               OUT errorcode INTEGER)
```

Para llamar a este procedimiento almacenado, tiene que colocar un valor CHAR para el parámetro IN, `job`, y un signo de interrogación, '?', para cada uno de los parámetros OUT. La sentencia SELECT del procedimiento utiliza el valor `job` de la columna JOB de la tabla EMPLOYEE, por ello, a fin de obtener una salida significativa, el parámetro IN tiene que ser un valor correcto de la columna JOB. El programa de ejemplo en C, `spclient`,

que llama al procedimiento almacenado, utiliza 'DESIGNER' para el valor de JOB. Se puede hacer lo mismo, del modo siguiente:

```
db2 "call MAIN_EXAMPLE ('DESIGNER', ?, ?)"
```

Después de ejecutar el mandato anterior, tendría que recibir un resultado como el siguiente:

```
Value of output parameters
--------------------------
Parameter Name  : SALARY
Parameter Value : +2.37312500000000E+004

Parameter Name  : ERRORCODE
Parameter Value : 0

Return Status = 0
```

Un ERRORCODE con el valor de cero indica un resultado satisfactorio.

Llamada a procedimientos almacenados desde aplicaciones CLI

Las aplicaciones CLI invocan procedimientos almacenados ejecutando la sentencia SQL del procedimiento CALL. Este tema describe cómo llamar a procedimientos almacenados desde aplicaciones CLI.

Antes de llamar a un procedimiento almacenado, asegúrese de que ha inicializado la aplicación CLI.

Si el procedimiento almacenado llamado no está catalogado, asegúrese de que no llama a ninguna de las funciones de esquema de CLI. La acción de llamar a las funciones de esquema de CLI desde procedimientos almacenados sin catalogar no está soportada.

Las funciones de esquema de CLI son: SQLColumns(), SQLColumnPrivileges(), SQLForeignKeys(), SQLPrimaryKeys(), SQLProcedureColumns(), SQLProcedures(), SQLSpecialColumns(), SQLStatistics(), SQLTables() y SQLTablePrivileges().

Para llamar a un procedimiento almacenado:

1. Declare las variables de sistema principal que corresponden a cada uno de los parámetros IN, INOUT y OUT del procedimiento almacenado. Asegúrese de que las longitudes y tipos de datos de variable de la aplicación se corresponden con las longitudes y tipos de datos de los argumentos de la signatura del procedimiento almacenado. DB2 CLI da soporte a la acción de llamar procedimientos almacenados con todos los tipos de SQL como marcadores de parámetro.
2. Inicialice las variables de aplicación de parámetro IN, INOUT y OUT.
3. Emita la sentencia CALL SQL. Por ejemplo:

   ```
   SQLCHAR *stmt = (SQLCHAR *)"CALL
   OUT_LANGUAGE (?)";
   ```

   Para un rendimiento óptimo, las aplicaciones deberían utilizar marcadores de parámetro para los argumentos de procedimiento almacenados en la serie del procedimiento CALL y después vincular las variables de sistema principal con dichos marcadores de parámetros. Sin embargo, si los argumentos de procedimiento almacenado de entrada deben especificarse como literales de serie en vez de como marcadores de parámetro, incluya los delimitadores de cláusula de escape de la llamada a ODBC { } en la sentencia de procedimiento CALL. Por ejemplo:

```
  SQLCHAR *stmt = (SQLCHAR
*)"{CALL IN_PARAM (123, 'Hello World!')}";
```

Cuando los literales de serie y la cláusula de escape de la llamada a ODBC se utilicen en una sentencia de procedimiento CALL, los literales de serie sólo podrán especificarse como argumentos de procedimiento almacenado de modalidad IN. Los argumentos de procedimiento almacenado de modalidad INOUT y OUT deberán seguir especificándose utilizando marcadores de parámetro.

4. Opcional: Prepare la sentencia CALL llamando a `SQLPrepare()`.
5. Vincule cada uno de los parámetros del procedimiento CALL llamando a `SQLBindParameter()`.

   Nota: Asegúrese de que cada uno de los parámetros se haya vinculado correctamente (a SQL_PARAM_INPUT, SQL_PARAM_OUTPUT o SQL_PARAM_INPUT_OUTPUT), en caso contrario podrían producirse resultados inesperados al ejecutarse la sentencia de procedimiento CALL. Esto se produciría, por ejemplo, en el caso de que un parámetro de entrada se hubiera vinculado incorrectamente con un *InputOutputType* de SQL_PARAM_OUTPUT.
6. Ejecute la sentencia del procedimiento CALL utilizando `SQLExecDirect()`, o en el caso de que la sentencia del procedimiento CALL se hubiera preparado en el paso 4, `SQLExecute()`.

   Nota: Si se termina una aplicación o hebra que hubiera invocado un procedimiento almacenado antes de que se hubiera completado el procedimiento almacenado, también terminará la ejecución del procedimiento almacenado. Es importante que un procedimiento almacenado contenga la lógica para asegurarse de que la base de datos esté en un estado coherente y deseable en el caso de que el procedimiento almacenado termine prematuramente.
7. Compruebe el código de retorno de `SQLExecDirect()` o `SQLExecute()` cuando se haya devuelto la función para determinar si se ha producido algún error durante la ejecución de la sentencia del procedimiento CALL o del procedimiento almacenado. Si el código de retorno es SQL_SUCCESS_WITH_INFO o SQL_ERROR, utilice las funciones de diagnóstico de CLI `SQLGetDiagRec()` y `SQLGetDiagField()` para determinar la razón por la que se produjo el error.

   Si un procedimiento almacenado se ha ejecutado satisfactoriamente, es posible que las variables vinculadas como parámetros OUT contengan datos que el procedimiento almacenado haya devuelto a la aplicación CLI. Si es de aplicación, es posible que el procedimiento almacenado también devuelva uno o más conjuntos de resultados por medio de cursores no desplazables. Las aplicaciones CLI deberían procesar conjuntos de resultados del procedimiento almacenado del mismo modo que deberían procesar los conjuntos de resultados generados mediante la ejecución de sentencias SELECT.

   Nota: Si una aplicación CLI no está segura del número o tipo de parámetros de un conjunto de resultados devuelto por un procedimiento almacenado, podrán llamarse las funciones `SQLNumResultCols()`, `SQLDescribeCol()` y `SQLColAttribute()` (por este orden) en el conjunto de resultados para determinar esta información.

Una vez haya ejecutado la sentencia CALL, podrá recuperar conjuntos de resultados del procedimiento almacenado si es de aplicación.

Nota:

Las partes numéricas del día y el mes de un valor tipo datos DATETYPE parecerán que están invertidas en los conjuntos de resultados del procedimiento que se devuelven a las aplicaciones DB2 CLI en el caso de que los valores no se devuelvan en formato ISO. Por ejemplo, esto podría producirse en el caso de que se utilice un formato local. Para asegurarse de que una aplicación cliente interprete correctamente la información de valor de tipo de datos DATETYPE, los procedimientos deberían vincularse a la base de datos con un formato DATETIME independiente del entorno local, como por ejemplo, ISO. Por ejemplo:

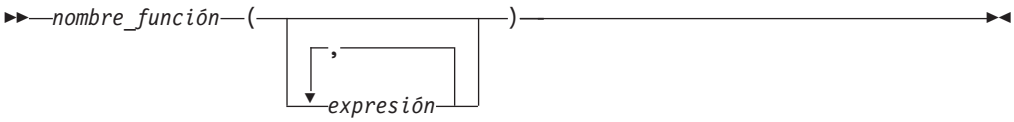
•
```
db2set DB2_SQLROUTINE_PREPOPTS="DATETIME ISO"
```

Nota:

Los paquetes de DB2 CLI se vinculan automáticamente a las bases de datos cuando éstas se crean o migran.

Referencias a funciones

Cada referencia a una función, tanto si es una UDF como una función incorporada, contiene la sintaxis siguiente:



En el diagrama de sintaxis anterior, `nombre_función` puede ser un nombre de función calificado o no calificado. Puede haber de 0 a 90 argumentos, que son expresiones. Ejemplos de componentes que pueden constituir expresiones son:

- un nombre de columna, calificado o no calificado
- una constante
- una variable del lenguaje principal
- un registro especial
- un marcador de parámetros

La posición de los argumentos es importante y se debe ajustar a la definición de la función para que la semántica sea correcta. Tanto la posición de los argumentos como la definición de la función se deben ajustar al propio cuerpo de la función. DB2 no intenta cambiar de sitio los argumentos para que coincidan mejor con una definición de función y DB2 no comprende la semántica de los parámetros de función individuales.

El uso de nombres de columna en las expresiones de argumento de UDF requiere que las referencias a las tablas que incluyen las columnas tengan el ámbito adecuado. Para las funciones de tabla referidas en una unión y que utilizan cualquier argumento que implique columnas de otra tabla o función de tabla, la tabla o función de tabla referida debe preceder a la función de tabla que contiene la referencia en la cláusula FROM.

Para poder utilizar marcadores de parámetros en funciones, no basta con codificar lo siguiente:

```
BLOOP(?)
```

Puesto que la lógica de selección de función no sabe en qué tipo de datos se puede convertir el argumento, no puede resolver la referencia. Se puede utilizar la especificación CAST para proporcionar el tipo del marcador de parámetros. Por ejemplo, INTEGER, y, luego, la lógica de selección de función podrá continuar:

```
BLOOP(CAST(? AS INTEGER))
```

He aquí algunos ejemplos válidos de invocaciones a funciones:

```
AVG(FLOAT_COLUMN)
BLOOP(COLUMN1)
BLOOP(FLOAT_COLUMN + CAST(? AS INTEGER))
BLOOP(:hostvar :indicvar)
BRIAN.PARSE(CHAR_COLUMN CONCAT USER, 1, 0, 0, 1)
CTR()
FLOOR(FLOAT_COLUMN)
PABLO.BLOOP(A+B)
PABLO.BLOOP(:hostvar)
"search_schema"(CURRENT FUNCTION PATH, 'GENE')
SUBSTR(COLUMN2,8,3)
SYSFUN.FLOOR(AVG(EMP.SALARY))
SYSFUN.AVG(SYSFUN.FLOOR(EMP.SALARY))
SYSIBM.SUBSTR(COLUMN2,11,LENGTH(COLUMN3))
SQRT((SELECT SUM(length*length)
     FROM triangles
     WHERE id= 'J522'
     AND legtype <> 'HYP'))
```

Si algunas de las funciones anteriores son funciones de tabla, la sintaxis para hacerles referencia es ligeramente distinta de la presentada antes. Por ejemplo, si PABLO.BLOOP es una función de tabla, para hacer referencia a ella de forma correcta, debe utilizar:

```
TABLE(PABLO.BLOOP(A+B)) AS Q
```

Selección de función

Para las referencias a funciones, tanto calificadas como no calificadas, el algoritmo de selección de función examina todas las funciones *pertinentes*, tanto incorporadas como definidas por el usuario, que tienen:

- El nombre indicado
- El mismo número de parámetros definidos que argumentos tiene la referencia a la función
- Cada uno de los parámetros idéntico o deducible a partir del tipo del argumento correspondiente.

Las funciones pertinentes son las funciones del esquema mencionado para una referencia calificada o las funciones de los esquemas de la vía de acceso SQL para una referencia no calificada. El algoritmo busca una coincidencia exacta o, en caso de no hallarla, la que más coincida entre estas funciones. La vía de acceso SQL se utiliza, únicamente en caso de una referencia no calificada, como factor decisorio si se encuentran dos coincidencias correctas idénticas en distintos esquemas.

Excepción: Si hay una referencia no calificada a una función denominada RID y se invoca la función con un solo argumento que coincide con una referencia de tabla en la cláusula ROM de la subselección, el esquema es SYSIBM y se invoca la función RID incorporada.

Es posible anidar las referencias a funciones, incluso referencias a la misma función. Generalmente sucede así con las funciones incorporadas y con las UDF; sin embargo, existen algunas limitaciones cuando se ven implicadas funciones de columna.

Por ejemplo:

```
CREATE FUNCTION BLOOP (INTEGER) RETURNS INTEGER ...
CREATE FUNCTION BLOOP (DOUBLE) RETURNS INTEGER ...
```

Considere ahora la sentencia DML siguiente:

```
SELECT BLOOP( BLOOP(COLUMN1)) FROM T
```

Si column1 es una columna DECIMAL o DOUBLE, la referencia BLOOP interior se resuelve en la segunda BLOOP definida más arriba. Dado que esta BLOOP devuelve un INTEGER, la BLOOP exterior se resuelve en la primera BLOOP.

Alternativamente, si column1 es una columna SMALLINT o INTEGER, la referencia BLOOP interior se resuelve en la primera BLOOP definida más arriba. Dado que esta BLOOP devuelve un INTEGER, la BLOOP exterior también se resuelve en la primera BLOOP. En este caso, está contemplando referencias anidadas a la misma función.

Definiendo una función con el nombre de uno de los operadores de SQL, realmente se puede invocar una UDF utilizando *notación infija*. Por ejemplo, suponga que puede adjudicar algún significado al operador "+" para los valores que tienen el tipo diferenciado BOAT. Puede definir la UDF siguiente:

```
CREATE FUNCTION "+" (BOAT, BOAT) RETURNS ...
```

A continuación, puede escribir la sentencia válida de SQL siguiente:

```
SELECT BOAT_COL1 + BOAT_COL2
FROM BIG_BOATS
WHERE BOAT_OWNER = 'Nelson Mattos'
```

Pero también puede escribir la sentencia siguiente, igualmente válida:

```
SELECT "+"(BOAT_COL1, BOAT_COL2)
FROM BIG_BOATS
WHERE BOAT_OWNER = 'Nelson Mattos'
```

Observe que no tiene permitido sobrecargar los operadores condicionales incorporados, tales como >, =, LIKE, IN, etc., de esta manera.

Tipos diferenciados como parámetros de UDF o método

Las UDF y los métodos se pueden definir con tipos diferenciados como parámetros o como resultado. DB2 pasará el valor a la UDF o al método con el formato del tipo de datos fuente del tipo diferenciado.

Los valores de tipo diferenciado que se originan en una variable del lenguaje principal y que se utilizan como argumentos para una UDF que tiene su parámetro correspondiente definido como tipo diferenciado, **los debe difundir explícitamente el usuario al tipo diferenciado**. No existe ningún tipo de lenguaje principal para los tipos diferenciados. La tipificación firme de DB2 lo requiere; de lo contrario, los resultados pueden ser ambiguos. Tome en consideración el tipo diferenciado BOAT que está definido sobre un BLOB y también la UDF BOAT_COST definida del modo siguiente:

```
CREATE FUNCTION BOAT_COST (BOAT)
  RETURNS INTEGER
  ...
```

En el fragmento siguiente de una aplicación en lenguaje C, la variable del lenguaje principal `:ship` contiene el valor BLOB que se debe pasar a la función BOAT_COST:

```
EXEC SQL BEGIN DECLARE SECTION;
  SQL TYPE IS BLOB(150K) ship;
EXEC SQL END DECLARE SECTION;
```

Las dos sentencias siguientes se resuelven correctamente en la función BOAT_COST, porque ambas difunden la variable del lenguaje principal `:ship` en el tipo BOAT:

```
... SELECT BOAT_COST (BOAT(:ship)) FROM ...
... SELECT BOAT_COST (CAST(:ship AS BOAT)) FROM ...
```

Si existen varios tipos diferenciados BOAT en la base de datos o varias UDF BOAT en otro esquema, debe tener cuidado con la vía de acceso SQL. De lo contrario, los resultados pueden ser ambiguos.

Valores de LOB como parámetros de UDF

Las UDF se pueden definir con parámetros o resultados que tengan cualquiera de los tipos de LOB: BLOB, CLOB o DBCLOB. DB2 materializará el valor del LOB entero en el almacenamiento antes de invocar una función de este tipo, aunque la fuente del valor sea una variable del lenguaje principal de *localizador de LOB*. Por ejemplo, considere el fragmento siguiente de una aplicación en lenguaje C:

```
EXEC SQL BEGIN DECLARE SECTION;
  SQL TYPE IS CLOB(150K) clob150K ;       /* var. leng. princ. LOB */
  SQL TYPE IS CLOB_LOCATOR clob_locator1; /* var. leng. princ. localizador LOB */
  char                   string[40];      /* var. leng. princ. serie */
EXEC SQL END DECLARE SECTION;
```

Cualquiera de las variables del lenguaje principal `:clob150K` o `:clob_locator1` son válidas como argumento para una función cuyo parámetro correspondiente esté definido como `CLOB(500K)`. Por ejemplo, suponga que ha registrado una UDF del modo siguiente:

```
CREATE FUNCTION FINDSTRING (CLOB(500K, VARCHAR(200)))
  ...
```

En el programa son válidas las dos invocaciones de FINDSTRING siguientes:

```
... SELECT FINDSTRING (:clob150K, :string) FROM ...
... SELECT FINDSTRING (:clob_locator1, :string) FROM ...
```

Los resultados o parámetros de UDF que tienen uno de los tipos de LOB se pueden crear con el modificador AS LOCATOR. En este caso, el valor de LOB entero no se materializa antes de la invocación. En cambio, se pasa un LOB LOCATOR a la UDF, la cual después puede utilizar SQL para manipular los bytes reales del valor de LOB.
También puede utilizar esta posibilidad en los resultados o parámetros de UDF que tienen un tipo diferenciado basado en un LOB. Observe que un argumento para una función de este tipo puede ser cualquier valor de LOB del tipo definido; no es necesario que sea una variable del lenguaje principal definida con uno de los tipos de LOCATOR. El uso de localizadores de variables del lenguaje principal como argumentos es completamente ortogonal al uso de AS LOCATOR en definiciones de resultados y parámetros de UDF.

Invocación de funciones escalares o métodos

La invocación de funciones escalares incorporadas, funciones escalares definidas por el usuario y métodos es muy similar. Las funciones escalares y los métodos sólo se pueden invocar donde estén soportadas las expresiones dentro de una sentencia de SQL.

Requisitos previos

- Para las funciones incorporadas, SYSIBM se debe encontrar en el registro especial CURRENT PATH. SYSIBM está en CURRENT PATH por omisión.
- Para las funciones escalares definidas por el usuario, la función se tiene que haber creado en la base de datos utilizando la sentencia CREATE FUNCTION o CREATE METHOD.
- Para las funciones escalares externas definidas por el usuario, el archivo de biblioteca o de clases asociado con la función se debe encontrar en la ubicación especificada por la cláusula EXTERNAL de la sentencia CREATE FUNCTION o CREATE METHOD.
- Para invocar una función definida por el usuario o método, un usuario debe tener el privilegio EXECUTE sobre la función o método. Si todos los usuarios van a utilizar la función o el método, el privilegio EXECUTE sobre la función o el método se puede otorgar a PUBLIC. Para obtener más información relativa a los privilegios, consulte la referencia a la sentencia CREATE específica.

Para invocar una UDF escalar o un método:

Incluya una referencia a la función o método dentro de una expresión contenida en una sentencia de SQL, donde se van a procesar uno o más valores de entrada. Las funciones y métodos se pueden invocar en cualquier parte en la que una expresión sea válida. Los ejemplos de las partes en las que se puede hacer referencia a una UDF escalar o método incluyen la lista de selección de una consulta o una cláusula VALUES.

Por ejemplo, suponga que ha creado una función escalar definida por el usuario denominada TOTAL_SAL que añade el salario base y la bonificación juntos para cada fila de empleado en la tabla EMPLOYEE.

```
CREATE FUNCTION TOTAL_SAL
  (SALARY DECIMAL(9,2), BONUS DECIMAL(9,2))
  RETURNS DECIMAL(9,2)
  LANGUAGE SQL
  CONTAINS SQL
  NO EXTERNAL ACTION
  DETERMINISTIC
  RETURN SALARY+BONUS
```

A continuación mostramos una sentencia SELECT que hace uso de TOTAL_SAL:

```
SELECT LASTNAME, TOTAL_SAL(SALARY, BONUS) AS TOTAL
  FROM EMPLOYEE
```

Invocación de funciones de tabla definidas por el usuario

Una vez que se escribe una función de tabla definida por el usuario y se registra con la base de datos, se puede invocar en la cláusula FROM de una sentencia SELECT.

Antes de comenzar

- La función de tabla se debe haber creado en la base de datos ejecutando CREATE FUNCTION.

- Para las funciones de tabla externas definidas por el usuario, el archivo de biblioteca o clases asociado con la función debe estar en la ubicación especificada por la cláusula EXTERNAL de CREATE FUNCTION.
- Para invocar una función de tabla definida por el usuario, un usuario debe tener el privilegio EXECUTE sobre la función. Si desea más información relacionada con los privilegios, consulte la referencia a CREATE FUNCTION.

Procedimiento

Para invocar una función de tabla definida por el usuario, haga referencia a la función en la cláusula FROM de una sentencia de SQL en la que se deba procesar un conjunto de valores de entrada. La referencia a la función de tabla debe ir precedida de la cláusula TABLE y debe aparecer entre corchetes.
Por ejemplo, la sentencia CREATE FUNCTION siguiente define una función de tabla que devuelve los empleados de un número de departamento especificado.

```
CREATE FUNCTION DEPTEMPLOYEES (DEPTNO VARCHAR(3))
  RETURNS TABLE (EMPNO CHAR(6),
                 LASTNAME VARCHAR(15),
                 FIRSTNAME VARCHAR(12))
  LANGUAGE SQL
  READS SQL DATA
  NO EXTERNAL ACTION
  DETERMINISTIC
  RETURN
    SELECT EMPNO, LASTNAME, FIRSTNME FROM EMPLOYEE
      WHERE EMPLOYEE.WORKDEPT = DEPTEMPLOYEES.DEPTNO
```

A continuación, se muestra una sentencia SELECT que hace uso de DEPTEMPLOYEES:

```
SELECT EMPNO, LASTNAME, FIRSTNAME FROM TABLE(DEPTEMPLOYEES('A00')) AS D
```

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.

Si desea ayudar a IBM para que los productos IBM Information Management sean más fáciles de utilizar, obtenga el cuestionario de consumibilidad: http://www.ibm.com/software/data/info/consumability-survey/.

© Copyright IBM Corp. 1993, 2010

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/e-business/linkweb/publications/servlet/pbi.wss. Los manuales de DB2 Versión 9.5 en inglés y las versiones traducidas en formato PDF se pueden descargar de www.ibm.com/support/docview.wss?uid=swg27009727 y www.ibm.com/support/docview.wss?uid=swg27009728 respectivamente.

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 en la tabla siguiente:

Nota: El *Centro de información de DB2* se actualiza con más frecuencia que los manuales en PDF o impresos.

Tabla 47. Información técnica de DB2

Nombre	Número de documento	Copia impresa disponible	Última actualización
Consulta de las API administrativas	SC11-3505-03	Sí	Diciembre de 2010
Rutinas y vistas administrativas	SC11-3507-03	No	Diciembre de 2010
Call Level Interface Guide and Reference, Volume 1	SC23-5844-03	Sí	Diciembre de 2010
Call Level Interface Guide and Reference, Volume 2	SC23-5845-03	Sí	Diciembre de 2010
Consulta de mandatos	SC11-3506-03	Sí	Diciembre de 2010
Data Movement Utilities Guide and Reference	SC23-5847-03	Sí	Diciembre de 2010
Data Recovery and High Availability Guide and Reference	SC23-5848-03	Sí	Diciembre de 2010
Data Servers, Databases, and Database Objects Guide	SC23-5849-03	Sí	Diciembre de 2010
Database Security Guide	SC23-5850-03	Sí	Diciembre de 2010
Developing ADO.NET and OLE DB Applications	SC11-3499-02	Sí	Abril de 2009
Developing Embedded SQL Applications	SC11-3500-02	Sí	Abril de 2009
Desarrollo de aplicaciones Java	SC11-3501-03	Sí	Diciembre de 2010
Desarrollo de aplicaciones Perl y PHP	SC11-3502-02	No	Abril de 2009

Tabla 47. Información técnica de DB2 (continuación)

Nombre	Número de documento	Copia impresa disponible	Última actualización
Desarrollo de rutinas definidas por el usuario (SQL y externas)	SC11-3503-03	Sí	Diciembre de 2010
Iniciación al desarrollo de aplicaciones de bases de datos	GC11-3504-03	Sí	Diciembre de 2010
Iniciación a la instalación y administración de DB2 en Linux y Windows	GC11-3511-03	Sí	Diciembre de 2010
Internationalization Guide	SC23-5858-02	Sí	Abril de 2009
Consulta de mensajes, Volumen 1	GI11-7823-00	No	
Consulta de mensajes, Volumen 2	GI11-7824-00	No	
Guía de migración	GC11-3509-03	Sí	Diciembre de 2010
Net Search Extender Guía de administración y del usuario	SC11-3615-02	Sí	Abril de 2009
Partitioning and Clustering Guide	SC23-5860-03	Sí	Diciembre de 2010
Query Patroller Administration and User's Guide	SC23-8507-01	Sí	Abril de 2009
Guía rápida para clientes de IBM Data Server	GC11-3513-03	No	Diciembre de 2010
Guía rápida para servidores DB2	GC11-3512-03	Sí	Diciembre de 2010
Spatial Extender and Geodetic Data Management Feature Guía del usuario y manual de consulta	SC11-3614-02	Sí	Abril de 2009
Consulta de SQL, Volumen 1	SC11-3509-03	Sí	Diciembre de 2010
Consulta de SQL, Volumen 2	SC11-3508-03	Sí	Diciembre de 2010
System Monitor Guide and Reference	SC23-5865-03	Sí	Diciembre de 2010
Guía de Text Search	SC11-3717-03	Sí	Diciembre de 2010
Troubleshooting Guide	GI11-7857-03	No	Diciembre de 2010
Tuning Database Performance	SC23-5867-03	Sí	Diciembre de 2010
Guía de aprendizaje de Visual Explain	SC11-3518-00	No	
Novedades	SC11-3517-03	Sí	Diciembre de 2010

Tabla 47. Información técnica de DB2 (continuación)

Nombre	Número de documento	Copia impresa disponible	Última actualización
Workload Manager Guide and Reference	SC23-5870-03	Sí	Diciembre de 2010
pureXML Guide	SC23-5871-03	Sí	Diciembre de 2010
XQuery Reference	SC23-5872-02	No	Abril de 2009

Tabla 48. Información técnica específica de DB2 Connect

Nombre	Número de documento	Copia impresa disponible	Última actualización
Guía rápida para DB2 Connect Personal Edition	GC11-3515-03	Sí	Diciembre de 2010
Guía rápida para servidores DB2 Connect	GC11-3516-03	Sí	Diciembre de 2010
Guía del usuario de DB2 Connect	SC11-3514-03	Sí	Diciembre de 2010

Tabla 49. Información técnica de Information Integration

Nombre	Número de documento	Copia impresa disponible	Última actualización
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	
Information Integration: Introduction to Replication and Event Publishing	GC19-1028-01	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 380.

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 Versión 9.8, el URL del *Centro de información de DB2* es http://publib.boulder.ibm.com/infocenter/db2luw/v9r8/.

Para los temas de la versión 9.7 de DB2, el URL del *Centro de información de DB2* es http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/.

Para los temas de DB2 Versión 9.5, el URL del *Centro de información de DB2* es http://publib.boulder.ibm.com/infocenter/db2luw/v9r5.

Para los temas de DB2 Versión 9.1, el URL del *Centro de información de DB2* es http://publib.boulder.ibm.com/infocenter/db2luw/v9/.

Para los temas de DB2 Versión 8, vaya al URL del *Centro de información de DB2* en: http://publib.boulder.ibm.com/infocenter/db2luw/v8/.

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 característica Actualizar para ver qué actualizaciones están disponibles. Si hay actualizaciones que desee instalar, puede utilizar la característica Actualizar para obtenerlos e instalarlos.

   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 *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 manera predeterminada, el *Centro de información de DB2*

está instalado en el directorio *Archivos_de_programa*\IBM\DB2 Information Center\Version 9.5, donde *Archivos_de_programa* representa la ubicación del directorio Archivos de programa.

c. Navegue desde el directorio de instalación al directorio doc\bin.

d. Ejecute el archivo help_start.bat:

```
help_start.bat
```

- En Linux:

a. Navegue hasta la vía de acceso en la que está instalado el Centro de información. De forma predeterminada, el *Centro de información de DB2* se instala en el directorio /opt/ibm/db2ic/V9.5.

b. Navegue desde el directorio de instalación al directorio doc/bin.

c. Ejecute el script help_start:

```
help_start
```

Se inicia el navegador Web por omisión de los sistemas para visualizar el Centro de información autónomo.

3. Pulse en el botón **Actualizar** ( ). En la derecha del panel del Centro de información, pulse en **Buscar actualizaciones**. Se visualiza una lista de actualizaciones para la documentación existente.
4. Para iniciar el proceso de 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 de base de datos DB2.

Documentación de DB2
Puede encontrar información sobre la resolución de problemas en la publicación DB2 Troubleshooting Guide o en la sección Conceptos fundamentales sobre bases de datos 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 la dirección http://www.ibm.com/software/data/db2/support/db2_9/

Términos y condiciones

Los permisos para utilizar estas publicaciones se otorgan sujetos a los siguientes términos y condiciones.

Uso personal: Puede reproducir estas publicaciones para su uso personal, no comercial, siempre y cuando se mantengan los avisos sobre la propiedad. No puede distribuir, visualizar o realizar trabajos derivados de estas publicaciones, o de partes de las mismas, sin el consentimiento expreso de IBM.

Uso comercial: Puede reproducir, distribuir y visualizar estas publicaciones únicamente dentro de su empresa, siempre y cuando se mantengan todos los avisos sobre la propiedad. No puede realizar trabajos derivativos de estas publicaciones, ni reproducirlas, distribuirlas o visualizarlas, ni de partes de las mismas fuera de su empresa, sin el consentimiento expreso de IBM.

Excepto lo expresamente concedido en este permiso, no se conceden otros permisos, licencias ni derechos, explícitos o implícitos, sobre las publicaciones ni sobre ninguna información, datos, software u otra propiedad intelectual contenida en el mismo.

IBM se reserva el derecho de retirar los permisos aquí concedidos cuando, a su discreción, el uso de las publicaciones sea en detrimento de su interés o cuando, según determine IBM, las instrucciones anteriores no se cumplan correctamente.

No puede descargar, exportar ni volver a exportar esta información excepto en el caso de cumplimiento total con todas las leyes y regulaciones vigentes, incluyendo todas las leyes y regulaciones sobre exportación de los Estados Unidos.

IBM NO GARANTIZA EL CONTENIDO DE ESTAS PUBLICACIONES. LAS PUBLICACIONES SE PROPORCIONAN "TAL CUAL" Y SIN GARANTÍA DE NINGUNA CLASE, NI EXPLÍCITA NI IMPLÍCITA, INCLUYENDO PERO SIN LIMITARSE A LAS GARANTÍAS IMPLÍCITAS DE COMERCIALIZACIÓN, NO VULNERACIÓN E IDONEIDAD PARA UN FIN DETERMINADO.

Apéndice B. Avisos

Esta información ha sido desarrollada para productos y servicios que se ofrecen en Estados Unidos de América

Es posible que IBM no comercialice en otros países algunos productos, servicios o características descritos en este manual. Consulte al representante local de IBM para obtener información sobre los productos y servicios que actualmente pueden adquirirse en su zona. Cualquier referencia a un producto, programa o servicio de IBM no pretende afirmar ni implicar que sólo se pueda utilizar dicho producto, programa o servicio de IBM. En su lugar se puede utilizar cualquier producto, programa o servicio funcionalmente equivalente que no vulnere ninguno de los derechos de propiedad intelectual de IBM. Sin embargo, es responsabilidad del usuario evaluar y verificar el funcionamiento de cualquier producto, programa o servicio que no sea de IBM.

IBM puede tener patentes o solicitudes de patentes en tramitación que afecten al tema tratado en este documento. La posesión de este documento no confiere ninguna licencia sobre dichas patentes. Puede realizar consultas sobre licencias escribiendo a:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
EE.UU.

Para realizar consultas sobre licencias referentes a información de doble byte (DBCS), puede ponerse en contacto con el Departamento de Propiedad Intelectual de IBM de su país/región o escribir a:

IBM World Trade Asia Corporation
Licensing
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japón

El párrafo siguiente no es aplicable al Reino Unido ni a ningún país/región en donde tales disposiciones sean incompatibles con la legislación local:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROPORCIONA ESTA PUBLICACIÓN "TAL CUAL", SIN GARANTÍA DE NINGUNA CLASE, NI EXPLÍCITA NI IMPLÍCITA, INCLUIDAS, PERO SIN LIMITARSE A ELLAS, LAS GARANTÍAS IMPLÍCITAS DE NO VULNERACIÓN DE DERECHOS, COMERCIALIZACIÓN O IDONEIDAD PARA UN FIN DETERMINADO. Algunos estados no permiten la exclusión de garantías expresas o implícitas en determinadas transacciones, por lo que es posible que esta declaración no sea aplicable en su caso.

Esta publicación puede contener inexactitudes técnicas o errores tipográficos. Periódicamente se efectúan cambios en la información aquí contenida; dichos cambios se incorporarán a las nuevas ediciones de la publicación. IBM puede efectuar, en cualquier momento y sin previo aviso, mejoras y cambios en los productos y programas descritos en esta publicación.

© Copyright IBM Corp. 1993, 2010

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
U59/3600
3600 Steeles Avenue East
Markham, Ontario L3R 9Z7
CANADÁ

Dicha información puede estar disponible, sujeta a los términos y condiciones apropiados, incluido en algunos casos el pago de una tarifa.

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

IBM, el logotipo de IBM e ibm.com son marcas registradas de International Business Machines Corp., que se han registrado en muchas otras jurisdicciones. Otros nombres de productos y servicios pueden ser marcas registradas de IBM o de otras empresas. Puede consultarse en línea una lista actualizada de las marcas registradas de IBM en la sección Copyright and trademark information de la web www.ibm.com/legal/copytrade.shtml.

Los siguientes términos son marcas registradas de otras empresas.

- Linux es una marca registrada de Linus Torvalds en los Estados Unidos y/o en otros países.
- Java y todas las marcas registradas basadas en Java son marcas registradas de Sun Microsystems, Inc. en los Estados Unidos y/o en otros países.
- UNIX es una marca registrada de The Open Group en los Estados Unidos y/o en otros países.
- Intel, el logotipo de Intel, Intel Inside, el logotipo de Intel Inside, Intel Centrino, el logotipo de Intel Centrino, Celeron, Intel Xeon, Intel SpeedStep, Itanium y Pentium son marcas registradas de Intel Corporation o de sus empresas subsidiarias en Estados Unidos y/o en otros países.
- Microsoft, Windows, Windows NT y el logotipo de Windows son marcas registradas de Microsoft Corporation 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

.NET
  Common Language Runtime
    ejemplo de rutina 203
    rutinas 153, 155, 167, 169
    soporte para el desarrollo de rutinas externas 154
  depuración de rutinas CLR 172
  rutinas
    opciones de compilación y enlace 171

A

a quién va dirigido este manual vii
actualizaciones
  Centro de información de DB2 385
ADMIN_CMD, procedimiento 33
AIX
  funciones definidas por el usuario en C++
    creación con archivos de configuración 284
  procedimientos almacenados en C++
    creación con archivos de configuración 282
  rutinas C
    opciones de compilación y enlace 271
  rutinas C++
    opciones de compilación y enlace 272
  rutinas COBOL de IBM
    crear 298
    opciones de compilación y enlace 293
  rutinas COBOL de Micro Focus
    opciones de compilación y enlace 293
ALLOCATE CURSOR, sentencia
  rutina llamadora 108
aplicaciones de varias hebras
  rutinas SQLJ 309
áreas reutilizables
  para UDF y métodos 145
  plataformas de 32 y 64 bits 148
  UDF Java 313
ASSOCIATE RESULT SET LOCATOR, sentencia 108
auditoría
  transacciones
    utilización de funciones de SQL 34
automatización de OLE
  identificador de clase (CLSID) 343
  identificador programático (progID) 343
  rutinas
    definición 343
    instancias de objetos 344
    invocación de métodos 344
    SCRATCHPAD, opción 344
  tipo de datos BSTR 346
  tipo de datos OLECHAR 346
  tipos de datos string 346
automatización OLE
  controladores 343
  métodos 343
  rutinas
    diseños 343
  servidores 343
autorizaciones
  para rutinas externas 57, 356
avisos 389
ayuda
  idioma de configuración 384
  sentencias SQL 383

B

BASIC, lenguaje 343
BASIC, tipos de datos 346
bibliotecas
  compartidas
    rutina de revinculación 284
BigDecimal, tipo de datos
  Java 307
BIGINT, tipo de datos
  COBOL 289
  función de tabla de OLE DB 352
  funciones definidas por el usuario (UDF) 236
  Java 307
  rutinas en Java (DB2GENERAL) 315
BLOB (objetos binarios grandes)
  COBOL 289
  función de tabla de OLE DB 352
  funciones definidas por el usuario (UDF)
    C/C++ 236
  Java 307
  rutinas
    Java (DB2GENERAL) 315
BLOB (objetos de caracteres grandes)
  tipo de datos
    COBOL 289
    función de tabla de OLE DB 352
    funciones definidas por el usuario (UDF) 236
    Java 307
    rutinas en Java (DB2GENERAL) 315
BLOB-FILE, tipo de COBOL 289
BLOB-LOCATOR, tipo de COBOL 289

C

C# .NET
  rutinas
    ejemplo 203
CALL, procedimientos 364
  desde activadores 366
  desde aplicaciones 365
  desde rutinas de SQL 366
  desde rutinas externas 365
CALL, sentencia
  Procesador de línea de mandatos 368
CAST FROM, cláusula
  manejo de tipos de datos 236
Centro de información de DB2
  actualización 385
  idiomas 384
  versiones 384
  visualización en distintos idiomas 384

© Copyright IBM Corp. 1993, 2010

CHAR, tipo de datos
COBOL 289
función de tabla de OLE DB 352
funciones definidas por el usuario (UDF)
C/C++ 236
Java 307
rutinas, Java (DB2GENERAL) 315
CHAR FOR BIT DATA, tipo de datos 315
CLASSPATH, variable de entorno 334
CLI (interfaz a nivel de llamada)
procedimientos almacenados
llamada 371
CLOB-FILE, tipo de COBOL 289
CLOB-LOCATOR, tipo de COBOL 289
CLP (procesador de línea de mandatos)
carácter de terminación 109
CLR (Common Language Runtime)
procedimientos
devolución de conjuntos de resultados 160
rutinas 153
consideraciones de diseño 155
creación 164, 165, 167, 169
ejemplos de procedimientos CLR en C# 176
ejemplos de UDF CLR en C# 210
herramientas de desarrollo 155
opciones de compilación y enlace 171
parámetros 158
restricciones 162
seguridad 161
soporte al desarrollo 154
soporte de XML 203
soporte de XQuery 203
COBOL, lenguaje
rutinas de Micro Focus
opciones de compilación en Solaris 295
software de desarrollo soportado para procedimientos externos 289
tipos de datos 289
COBOL, tipos de datos
BLOB 289
BLOB-FILE 289
BLOB-LOCATOR 289
CLOB 289
CLOB-FILE 289
CLOB-LOCATOR 289
COMP-1 289
COMP-3 289
COMP-5 289
DBCLOB 289
DBCLOB-FILE 289
DBCLOB-LOCATOR 289
PICTURE (PIC), cláusula 289
USAGE, cláusula 289
COM.ibm.db2.app.Blob 315, 320
COM.ibm.db2.app.Clob 315, 321
COM.ibm.db2.app.Lob 320
COM.ibm.db2.app.StoredProc 317
COM.ibm.db2.app.UDF 313, 318
coma flotante
parámetro 236
Common Language Runtime
ejemplos de función 187
ejemplos de procedimiento 192
procedimientos
devolución de conjuntos de resultados 160
rutinas 153
área-reutilizable 158
Common Language Runtime *(continuación)*
rutinas *(continuación)*
consideraciones de diseño 155
creación 164, 165, 167, 169
ejemplos 176, 187, 192
ejemplos de funciones CLR en C# 210
ejemplos de procedimientos CLR en C# 176
errores 173
herramientas de desarrollo 155
parámetros 158
restricciones 162
seguridad 161
soporte al desarrollo 154
tipos de datos de SQL soportados en 156
uso de la estructura Dbinfo 158
cómo está estructurado este manual vii
COMP-1, tipos de datos
COBOL 289
COMP-3, tipos de datos
COBOL 289
COMP-5, tipos de datos
COBOL 289
comparación
de los tipos funcionales de rutinas 16
compartidas, bibliotecas
rutina de revinculación 284
con fuente
rutinas 22
conjuntos de filas
OLE DB
nombres completamente calificados 351
conjuntos de resultados
devolución
procedimientos .NET CLR 160
procedimientos almacenados JDBC 322
procedimientos almacenados SQLJ 323
procedimientos de SQL 107
recepción en aplicaciones y rutinas JDBC 324
recepción en aplicaciones y rutinas SQLJ 325
recibir desde una rutina SQL 108
contextos
establecimiento en aplicaciones de DB2 de varias hebras
rutinas SQLJ 309
copia de seguridad
bibliotecas de rutinas externas 125
correlación de tipos
automatización de OLE
tipos BASIC 346
corto, tipo de datos
Java 307
creación
rutinas 63, 151, 261
Common Language Runtime 164, 165
Data Studio 37
crear
rutinas 6, 71
C/C++ 218
Data Studio 66
herramientas de desarrollo para SQL 66
CREATE FUNCTION, sentencia
CAST FROM, cláusula 236
LANGUAGE OLE, cláusula 343
PARAMETER STYLE, cláusula 225, 312
RETURNS, cláusula 236
rutinas de automatización de OLE 343
CREATE METHOD, sentencia
PARAMETER STYLE, cláusula 312

CREATE PROCEDURE, sentencia
cláusula PROGRAM TYPE 231
con procedimientos de SQL 109
PARAMETER STYLE, cláusula 222, 310
cursores
procedimientos de SQL 95
XQuery 96

D

Data Studio
soporte de rutinas SQL 66
visión general 37
DATE, tipo de datos
COBOL 289
función de tabla de OLE DB 352
Java 307
rutinas DB2GENERAL 315
DB2_USE_DB2JCCT2_JROUTINE, variable
especificación de controladores 306
DB2GENERAL, estilo de parámetro 133
DB2SQL, estilo de parámetro para rutinas externas 133
DBCLOB, tipo de datos
COBOL 289
función de tabla de OLE DB 352
funciones definidas por el usuario (UDF)
C/C++ 236
Java 307
rutinas
Java (DB2GENERAL) 315
DBCLOB-FILE COBOL, tipo de datos 289
DBCLOB-LOCATOR COBOL, tipo de datos 289
dbinfo, argumento
funciones de tabla 142
DBINFO, opción
páginas de códigos 362
DECIMAL, tipo de datos
conversión
COBOL 289
Java 307
función de tabla de OLE DB 352
funciones definidas por el usuario (UDF) 236
rutinas DB2GENERAL 315
DECLARE, sentencias
procedimientos de SQL
condiciones 98
cursores 98
descriptor de contexto de condición 98
variables 91, 98
decoración de tipos
cuerpos de rutinas en C++ 257
definidas por el sistema
rutinas 5
depurar
rutinas
CLR .NET 172
desarrollo de aplicaciones
rutinas 2
determinación de problemas
guías de aprendizaje 387
información disponible 387
devolución de conjuntos de resultados
desde procedimientos almacenados JDBC 322
desde procedimientos almacenados SQLJ 323
documentación
copia impresa 380
PDF 380
documentación *(continuación)*
términos y condiciones de uso 387
visión general 379
DOUBLE, tipo de datos
funciones definidas por el usuario (UDF)
C/C++ 236
Java 307

E

errores
rutinas .NET CLR 173
estilos de parámetros
PARAMETER STYLE DB2GENERAL 310
PARAMETER STYLE JAVA 310
visión general 133
EXECUTE, privilegio
rutinas 57, 356

F

FLOAT, tipo de datos
COBOL 289
función de tabla de OLE DB 352
funciones definidas por el usuario (UDF)
C/C++ 236
Java 307
rutinas Java (DB2GENERAL) 315
funciones
algoritmo de selección 374
comparación
con otros tipos funcionales de rutinas 16
externas
características 139
funciones de fila 14
funciones de tabla 14
funciones escalares 13
invocar 373
Java
PARAMETER STYLE, cláusula 312
parámetros
PARAMETER STYLE, cláusula 225
referencias de sintaxis 373
selección 374
visión general 11
visión general de rutinas 32
funciones de fila 14
funciones de SQL
comparación con los procedimientos de SQL 73
funciones de tabla
que modifican datos de SQL 34
soporte de sentencias de SQL 38
funciones de tabla 14
funciones de tabla definidas por el usuario 142
modelo de ejecución de Java 144, 326
funciones definidas por el usuario
archivos de configuración de C++ de AIX 284
C/C++
argumentos 236
BIGINT, tipo de datos 236
BLOB, tipo de datos 236
CHAR, tipo de datos 236
CLOB, tipo de datos 236
DBCLOB, tipo de datos 236
DOUBLE, tipo de datos 236
FLOAT, tipo de datos 236

funciones definidas por el usuario *(continuación)*
C/C++ *(continuación)*
INTEGER, tipo de datos 236
LONG VARCHAR, tipo de datos 236
parámetros 236
REAL, tipo de datos 236
SMALLINT, tipo de datos 236
VARCHAR FOR BIT DATA, tipo de datos 236
VARGRAPHIC, tipo de datos 236
de tabla 142
FINAL CALL 142
invocar 377
modelo de proceso 142
NO FINAL CALL 142
SQL-result, argumento 142
SQL-result-ind, argumento 142
DETERMINISTIC 145
devolución de datos 236
escalares
FINAL CALL 141
FOR BIT DATA, modificador 236
funciones de tabla de OLE DB 349
guardar el estado 145
infix, notación 374
invocar 377
Java
restricciones de E/S 313
NOT DETERMINISTIC 145
parámetros de fecha 236
portabilidad del área reutilizable entre plataformas de 32 y 64 bits 148
reentrantes 145
SCRATCHPAD, opción 145
tipos diferenciados como parámetros 375
UDF Common Language Runtime
ejemplos en C# 210
usos de 33
valores LOB como parámetros 376
volver a grabar procedimientos de SQL como UDF 75
funciones definidas por el usuario (UDF) para tablas
modelo de proceso 142
funciones escalares
descripción 139
modelo de proceso 141
rutinas 13

G

GENERAL, estilo de parámetro para rutinas externas 133
GENERAL WITH NULLS, estilo de parámetro para rutinas externas 133
GRAPHIC, parámetro 236
GRAPHIC, tipo de datos
COBOL 289
función de tabla de OLE DB 352
Java 307
rutinas
Java (DB2GENERAL) 315
guías de aprendizaje
determinación de problemas 387
resolución de problemas 387
Visual Explain 386

H

herramientas
desarrollo de rutinas 37
HP-UX
opciones de compilación
rutinas C 273
rutinas C++ 274
rutinas COBOL de Micro Focus 294
opciones de enlace
rutinas C 273
rutinas C++ 274
rutinas COBOL de Micro Focus 294

I

infix, notación
funciones definidas por el usuario (UDF) 374
instancia de objeto
rutinas de automatización de OLE 344
Int, tipo de datos de Java 307
INTEGER, tipo de datos
COBOL 289
función de tabla de OLE DB 352
funciones definidas por el usuario (UDF)
C/C++ 236
Java 307
rutinas
Java (DB2GENERAL) 315
invocar
funciones de tabla definidas por el usuario 377
procedimientos almacenados 364
rutinas 355
UDF 377

J

Java
archivos de clases 334
archivos JAR 336
clases 337
CLASSPATH, variable de entorno 334
funciones 312
métodos
COM.ibm.db2.app.Blob 320
COM.ibm.db2.app.Clob 321
COM.ibm.db2.app.Lob 320
COM.ibm.db2.app.StoredProc 317
COM.ibm.db2.app.UDF 313, 318
PARAMETER STYLE JAVA 312
modelo de ejecución de las funciones de tabla 144, 326
PARAMETER STYLE DB2GENERAL 313
PARAMETER STYLE JAVA 310, 312
procedimientos
PARAMETER STYLE JAVA 310
procedimientos almacenados
archivos JAR 336
visión general 303
rutinas
controladores 306
creación (JDBC) 331
creación (SQLJ) 333
creación (visión general) 331
diseño 307
estilos de parámetros 133, 310
herramientas de desarrollo 306
PARAMETER STYLE DB2GENERAL 313

Java *(continuación)*
rutinas *(continuación)*
restricciones 326
software de desarrollo soportado 304
UNIX 305
visión general 303
tipos de datos 307
UDF (funciones definidas por el usuario)
áreas reutilizables 313
DB2GENERAL 313
FENCED 313
NOT FENCED 313
sentencia CALL para archivos JAR 336
Java Database Connectivity (JDBC)
procedimientos almacenados 322
rutinas
API 304
controladores 304
creación 329
creación (visión general) 331
ejemplos (resumen) 337
ejemplos (utilizando el tipo de datos de matriz) 337
ejemplos (XML y soporte de XQuery) 338
herramientas de desarrollo 306
tipo de datos ARRAY 321, 337
XML
ejemplo 338
java.math.BigDecimal, tipo de datos de Java 307
JDBC
rutinas
creación (procedimiento) 331
jdk_path, parámetro de configuración
desarrollo de aplicaciones 285
rutinas
creación (UNIX) 305
ejecución (UNIX) 305

K

keepfenced, parámetro de configuración
actualización 285
Kit de desarrollo de software de IBM
desarrollo de rutinas Java 304

L

LANGUAGE OLE, cláusula
CREATE FUNCTION, sentencia 343
lenguaje C
funciones
estilos de parámetros 225
procedimientos
conjuntos de resultados 259
ejemplo 206
estilos de parámetros 222
soporte de XML 206
soporte de XQuery 206
rutinas 216
archivo de inclusión 219
área reutilizable como parámetro de función 230
cláusula PROGRAM TYPE 231
conjuntos de resultados 228
creación 261, 263, 269
creación en UNIX 265
creación en Windows 267
diseño 218
lenguaje C *(continuación)*
rutinas *(continuación)*
estilos de parámetros 221
estructura dbinfo como parámetro 228
herramientas de desarrollo 218
opciones de compilación en AIX 271
opciones de compilación en HP-UX 273
opciones de compilación en Linux 276
opciones de compilación en Solaris 279
opciones de compilación en Windows 282
parámetros 220
parámetros de indicadores nulos 221
pase de parámetros 227
rutinas de 32 bits en un servidor de bases de datos de 64 bits 132
sintaxis para pasar argumentos 244
soporte al desarrollo 217
tipos de datos de SQL soportados en 233
lenguaje C/C++
decoración de tipos para cuerpos de rutinas 257
funciones
estilos de parámetros 225
procedimientos
conjuntos de resultados 259
estilos de parámetros 222
rutinas 216
archivo de inclusión 219
área reutilizable como parámetro de función 230
cláusula PROGRAM TYPE 231
conjuntos de resultados 228
creación 261, 263, 269
creación en Windows 267
diseño 218
estilos de parámetros 221
estructura dbinfo como parámetro 228
herramientas de desarrollo 218
opciones de compilación en AIX 272
opciones de compilación en HP-UX 274
opciones de compilación en Linux 277
opciones de compilación en Solaris 280
opciones de compilación en Windows 282
parámetros 220
parámetros de indicadores nulos 221
pase de parámetros 227
rutinas de 32 bits en un servidor de bases de datos de 64 bits 132
soporte al desarrollo 218
tipos de datos de SQL soportados en 233
tipos de datos
automatización de OLE 346
lenguaje COBOL
procedimientos almacenados 287
rutinas COBOL de IBM
creación en AIX 298
creación en Windows 300
opciones de compilación en AIX 293
opciones de compilación en Windows 296
rutinas de Micro Focus
creación en UNIX 299
creación en Windows 301
opciones de compilación en AIX 293
opciones de compilación en HP-UX 294
opciones de compilación en Linux 296
opciones de compilación en Windows 297
lenguaje de procedimientos de SQL
comparación entre SQL PL y SQL PL incorporado 67
ejecución de SQL PL incorporado 69

lenguaje de procedimientos de SQL *(continuación)*
procedimientos de SQL 69
rendimiento 112
sentencias de flujo de control 98
SQL PL incorporado
ejecutar 69
visión general 66
Linux
rutinas C
opciones de compilación y enlace 276
rutinas C++
opciones de compilación y enlace 277
rutinas COBOL de Micro Focus
opciones de compilación y enlace 296
LOB (objetos grandes)
pase a rutinas 376
LONG VARCHAR, tipo de datos
COBOL 289
función de tabla de OLE DB 352
funciones definidas por el usuario (UDF)
C/C++ 236
Java 307
rutinas
Java (DB2GENERAL) 315
LONG VARCHAR FOR BIT DATA, tipo de datos
rutinas
Java (DB2GENERAL) 315
LONG VARGRAPHIC, tipo de datos
COBOL 289
función de tabla de OLE DB 352
Java 307
parámetro de las UDF 236
rutinas
Java (DB2GENERAL) 315

M

manejadores de condiciones
cláusula CONTINUE 93
procedimientos de SQL
descripción 107
manuales
copia impresa
pedido 382
marcadores de parámetros
ejemplos 86
visión general 86
métodos 15
comparación
con otros tipos funcionales de rutinas 16
externas
características 139
Java
PARAMETER STYLE, cláusula 312
tipos diferenciados como parámetros 375
métodos de SQL
soporte de sentencias de SQL 38
MODIFIES SQL DATA, cláusula
niveles de acceso a SQL en rutinas de SQL 43

N

NUMERIC, parámetro 236
NUMERIC, tipo de datos de SQL
COBOL 289
función de tabla de OLE DB 352
NUMERIC, tipo de datos de SQL *(continuación)*
Java 307
rutinas
Java (DB2GENERAL) 315

O

Object Linking and Embedding (OLE) 343
objetos binarios grandes (BLOB)
COBOL 289
función de tabla de OLE DB 352
funciones definidas por el usuario (UDF)
C/C++ 236
Java 307
rutinas
Java (DB2GENERAL) 315
objetos grandes (LOB)
pase a rutinas 376
OLE, rutinas de
sintaxis para pasar argumentos 244
OLE DB
funciones de tabla
conexión de serie en la cláusula EXTERNAL NAME 350
creación 350
definidas por el usuario 349
opción CONNECTSTRING 350
utilizando el nombre de servidor 350
nombres de fila completamente calificados 351
tipos de datos
conversión a tipos de datos de SQL 352

P

páginas de códigos
rutinas de conversión 362
parámetros
procedimientos de SQL 85
rutinas C/C++ 220
parámetros de configuración
javaheapsz, parámetro de configuración 334
jdk11path, parámetro de configuración 334
pedido de manuales de DB2 382
PICTURE (PIC), cláusula en tipos de COBOL 289
portabilidad
rutinas 45
precompilación
procedimientos de SQL 111
procedimientos
ADMIN_CMD
visión general 33
Common Language Runtime (CLR)
ejemplos 176
comparación de funciones 16
comparación de métodos 16
conjuntos de resultados
.NET CLR (procedimiento) 160
CLR de .NET (ejemplos en C#) 176
rutinas de SQL 108
conjuntos de resultados C/C++ 259
efectos de confirmación en parámetros y variables XML 95
efectos de retrotracción en parámetros y variables XML 95
Java
PARAMETER STYLE JAVA, cláusula 310

procedimientos *(continuación)*
llamada
desde activadores 366
desde aplicaciones 365
desde rutinas de SQL 366
desde rutinas externas 365
visión general 364
parámetros
cláusula PARAMETER STYLE SQL 222
PARAMETER STYLE JAVA, cláusula 310
referencias 363
SQL
características 79
componentes 80
diseño 80
estructura 81
manejadores de condiciones 107
parámetros 85
sentencias compuestas 94
sentencias condicionales 98
sentencias de flujo de control 98, 99, 100
sentencias de transferencia de control 103
sentencias en bucle 100
soporte a matrices 84
usos 79
variables 91, 94
visión general 79
visión general 10
XML
parámetros 92
variables 92
procedimientos almacenados 10
algoritmo de selección 364
archivos de configuración de C++ de AIX 282
CALL, sentencia 368
COBOL 287
invocar 364
llamada
aplicaciones CLI 371
referencias (sintaxis de referencias de llamada) 363
selección 364
procedimientos de SQL
ATOMIC, sentencias compuestas 81
CALL, sentencia 368
características 79
comparación con las funciones de SQL 73
comparación con SQL compuesto dinámico 74
componentes 80
creación 109
cursores 95
devolución de conjuntos de resultados 107
diseño 80
estructura 81
etiquetas 81
manejadores de condiciones 107
NOT ATOMIC, sentencias compuestas 81
opciones de vinculación y precompilación 111
parámetros 85
rendimiento 112
sentencias condicionales 98
sentencias de flujo de control 98, 99, 100
sentencias de transferencia de control 103
sentencia GOTO 104
sentencia ITERATE 105
sentencia LEAVE 105
sentencia RETURN 106
sentencias en bucle 100
procedimientos de SQL *(continuación)*
LOOP, sentencia 101
sentencia FOR 100
sentencia REPEAT 103
WHILE, sentencia 102
soporte de sentencias de SQL 38
usos 79
variables 91
variables SQLCODE y SQLSTATE 93
visión general 79
volver a grabar como UDF de SQL 75
procedimientos externos
COBOL 289
procesador de línea de mandatos (CLP)
creación de rutinas 37

R

REAL, tipo de datos de SQL
COBOL 289
conversión
en rutinas C y C++ 236
en rutinas Java (DB2GENERAL) 315
función de tabla de OLE DB 352
Java 307
recibir
conjuntos de resultados
desde una rutina SQL 108
en aplicaciones y rutinas JDBC 324
en aplicaciones y rutinas SQLJ 325
rendimiento
procedimientos de SQL 112
rutinas
beneficios 2
recomendaciones 46
rutinas externas 125
resolución de problemas
guías de aprendizaje 387
información en línea 387
restauración
bibliotecas de rutinas externas 125
restricciones
rutinas 135
RETURNS, cláusula
CREATE FUNCTION, sentencia 236
rutinas
administración de la base de datos 33
anidadas 361
automatización de OLE
definición 343
beneficios 2
bibliotecas 121
C/C++
archivo de inclusión 219
área reutilizable como parámetro de función 230
cláusula PROGRAM TYPE 231
conjuntos de resultados 228, 259
creación 261, 263, 269
descripción 216
diseño 218
estilos de parámetros 221
estructura sqludf_scrat 230
herramientas de desarrollo 218
parámetros 220, 228
parámetros de indicadores nulos 221
pase de parámetros 227
pase por referencia 227

rutinas *(continuación)*
C/C++ *(continuación)*
pase por valor 227
rendimiento 132
rutinas de 32 bits en un servidor de bases de datos de 64 bits 132
soporte al desarrollo 217, 218
soporte de tipo de datos xml 149
tipos de datos de SQL soportados en 233
variables gráficas del lenguaje principal 257
clases 121
CLR
errores 173
COBOL
soporte de tipo de datos xml 149
Common Language Runtime
cláusula EXECUTION CONTROL 161
consideraciones de diseño 155
creación 164, 167, 169
descripción 153
devolución de conjuntos de resultados 160
ejemplos 176
ejemplos de funciones (UDF) CLR 210
ejemplos de funciones CLR de .NET en Visual Basic 187
ejemplos de procedimientos CLR en C# 176
ejemplos de procedimientos Visual Basic .NET CLR 192
errores 173
herramientas de desarrollo 155
restricciones 162
seguridad 161
soporte al desarrollo 154
soporte de tipo de datos xml 149
tipos de datos de SQL soportados en 156
uso del área reutilizable 158
comparación 7, 16
de los tipos funcionales 16
definidas por el sistema y definidas por el usuario 8
SQL y externas 71
con fuente 21, 29, 32
conflictos de lectura 60
creación
Data Studio 37
crear
seguridad 54
DB2GENERAL
clases de Java 316
COM.ibm.db2.app.Blob 320
COM.ibm.db2.app.Clob 321
COM.ibm.db2.app.Lob 320
descripción 313
definición de la estructura del área reutilizable 148
definidas por el sistema 1, 3, 5, 7, 33
cuándo utilizar 8
definidas por el usuario 1, 3, 5, 6, 7, 33
creación 63
cuándo utilizar 8
determinar la implementación a utilizar 32
definidores de 5
definidos por el usuario 150
descripción 2
emisión de sentencias CREATE 109
escritura 150
conflictos 60
EXECUTE, privilegio 57, 356
externas 21, 29, 32
rutinas *(continuación)*
actualización de rutinas de Java 337
API y lenguajes de programación soportados 23, 125, 126
autorizaciones para 57, 356
C/C++ 216, 218, 263, 269
características 119, 139
Common Language Runtime 153, 164, 165, 167, 169
comparación con las de SQL 71
conflictos de denominación 123
copia de seguridad y restauración de archivos de bibliotecas y de clases 125
creación 151
crear 120
descripción 22
despliegue de bibliotecas y clases 122
estilos de parámetros 133
gestión de bibliotecas 125
Java 331
modificación de archivos de bibliotecas y clases 124
rendimiento 125
restricciones 135, 139
seguridad 123
sentencias prohibidas 135
soporte de 32 y 64 bits 131
soporte de sentencias de SQL 38
soporte de tipo de datos xml 149
visión general 1, 119
funciones 11
cuándo utilizar 20
funciones de fila 14
funciones de tabla 14
funciones escalares 13
herramientas de desarrollo 37
implementaciones 21
comparación 32
con fuente 22
incorporadas 21
SQL 22
incorporadas 21, 29
interoperatividad 46
invocar 355
desde otras rutinas 46
rutinas de 32 bits en un servidor de bases de datos de 64 bits 362
seguridad 54
Java 304
archivos JAR 336
creación 328, 329
JDBC 331
restricciones 326
soporte de tipo de datos xml 149
visión general 303
métodos 15, 150
cuándo utilizar 20
modificación 121
nombre 360
NOT FENCED
seguridad 54, 56
páginas de códigos
conversión 362
pase de LOB a 376
pase de tipos diferenciados a 375
portabilidad 45
portabilidad entre plataformas de 32 bits y de 64 bits 148
procedimientos 10, 150
cuándo utilizar 20

rutinas *(continuación)*
recepción de conjuntos de resultados 108
rendimiento 46
repetitivas 361
restricciones 135
revinculación de bibliotecas compartidas 284
seguridad 54, 56
sentencias de SQL aceptables 44
sentencias prohibidas 135
sintaxis para pasar argumentos 244
sobrecarga 360
SQL 21, 22, 29, 32
comparación con las externas 71
crear 71
herramientas de desarrollo 66
rendimiento 112
soporte de sentencias de SQL 38
visión general 65
tipos 1, 3, 20
comparación 32
comparación de los tipos funcionales 16
funcionales 9
sentencias de SQL soportadas 38
tipos funcionales de 9
UDF escalares
visión general 139
usos de 32, 33
variables gráficas del lenguaje principal 257
vía de acceso a la función 360
visión general 1
WCHARTYPE, opción de precompilador 257
rutinas CLR
.NET
depurar 172
rutinas con fuente 22
rutinas DB2GENERAL
clases de Java
COM.ibm.db2.app.Blob 320
COM.ibm.db2.app.Clob 321
COM.ibm.db2.app.Lob 320
COM.ibm.db2.app.StoredProc 317
COM.ibm.db2.app.UDF 318
visión general 316
funciones definidas por el usuario 313, 318
procedimientos almacenados 317
visión general 313
rutinas de SQL 22
comparación con las rutinas externas 71
crear 71
herramientas de desarrollo 66
Data Studio 66
visión general 65
rutinas definidas por el usuario 6
rutinas externas
API 23, 125, 126
archivos de clases
modificar 124
restaura 125
seguridad 123
bibliotecas
copia de seguridad 125
despliegue 122
gestión 125
modificar 124
rendimiento 125
restauración 125
seguridad 123
rutinas externas *(continuación)*
características 119
clases
despliegue 122
comparación de rutinas QL 71
conflictos de denominación 123
creación 151
crear 120
ejemplos 337
estilos de parámetros 133
lenguajes de programación 23, 125, 126
rendimiento 125
soporte de 32 bits 131
soporte de 64 bits 131
visión general 22, 119
rutinas incorporadas 21
rutinas SQL
visión general 65

S

SCRATCHPAD, opción
conservación del estado 145
funciones definidas por el usuario (UDF) 145
rutinas de automatización de OLE 344
SDK
Java
UNIX 305
seguridad
rutinas 54, 56
sentencia CASE 98, 99
sentencia CREATE ROUTINE
PARAMETER STYLE, cláusula 221
sentencia FOR 100
sentencia GOTO 104
sentencia IF 98, 100
sentencia ITERATE 105
sentencia LEAVE 105
sentencia LOOP
procedimientos de SQL 101
sentencia REPEAT 103
sentencia RETURN 106
sentencia WHILE
procedimientos de SQL 102
sentencias
CREATE FUNCTION 63, 343
CREATE METHOD 63
CREATE PROCEDURE 63
sentencias de SQL
permitidas en rutinas 38, 44
relacionados con variables 98
sentencias de transferencia de control 103
sentencias en bucle 100
sentencias SQL
visualización de la ayuda 383
sentencias XQuery
llamada desde SQL 96
SET, sentencia
establecer variables en procedimientos de SQL 91
sistemas operativos Solaris
rutinas
opciones de compilación y enlace en C 279
opciones de compilación y enlace en C++ 280
rutinas COBOL de Micro Focus
opciones de compilación y enlace 295

sistemas operativos Windows
rutinas C/C++
opciones de enlace 282
rutinas COBOL
opciones de enlace 296
rutinas COBOL de Micro Focus
opciones de enlace 297
Sistemas operativos Windows
rutinas C/C++
crear 267
opciones de compilación 282
rutinas COBOL
crear 300
opciones de compilación 296
rutinas COBOL de Micro Focus
crear 301
opciones de compilación 297
SMALLINT, tipo de datos
COBOL 289
función de tabla de OLE DB 352
funciones definidas por el usuario (UDF)
C/C++ 236
Java 307
rutinas
Java (DB2GENERAL) 315
sobrecarga
nombres de rutina 360
soporte de 32 bits
rutinas externas 131
soporte de 64 bits
rutinas externas 131
soporte de plataformas cruzadas
Invocación de rutinas de 32 bits en un servidor de bases de datos de 64 bits 362
SQL (Structured Query Language)
estilo de parámetro para rutinas externas 133
rutinas
niveles de acceso al SQL en rutinas incorporadas al SQL 43
SQL compuesto
procedimientos de SQL 94
SQL de transact 10
SQL dinámico
comparación de procedimientos de SQL 74
SQL-result, argumento
funciones de tabla 142
SQL-result-ind, argumento
funciones de tabla 142
SQLCODE
variables en procedimientos de SQL 93
sqldbchar, tipo de datos
en rutinas C/C++ 236
SQLJ
creación de rutinas 304, 333
procedimientos almacenados
devolución de conjuntos de resultados 323
rutinas
contextos de conexión 309
creación 328
herramientas de desarrollo 306
opciones del compilador (UNIX) 333
opciones del compilador (Windows) 334
SQLSTATE
variables en procedimientos de SQL 93
SQLUDF, archivo de inclusión
rutinas C/C++ 219
String, tipo de datos de Java 307

T

tablas
acceso
conflictos de lectura y grabación de rutinas 60
funciones de tabla 14
términos y condiciones
uso de publicaciones 387
TIME, parámetro 236
TIME, tipos de datos
COBOL 289
función de tabla de OLE DB 352
Java 307
rutinas Java 315
TIMESTAMP, parámetro 236
TIMESTAMP, tipo de datos
COBOL 289
función de tabla de OLE DB 352
Java 307
rutinas
Java (DB2GENERAL) 315
tipos de columna
creación
COBOL 289
tipos de datos
ARRAY 321
COBOL 289
conversión
COBOL 289
tipos de automatización de OLE 345
programas Java
visión general 307
tipos de datos de SQL
COBOL 289
conversión a tipos de datos de OLE DB 352
funciones definidas por el usuario (UDF)
C/C++ 236
Java 307
rutinas
Java (DB2GENERAL) 315
soportados en la automatización de OLE 345
tipos diferenciados
pase a rutinas 375
tipos estructurados
atributos
acceso a atributos con métodos 15
métodos 15

U

UNIX
rutinas C
crear 265
rutinas COBOL de Micro Focus
crear 299
rutinas SQLJ 333
USAGE, cláusula en tipos de COBOL 289

V

VARCHAR, tipo de datos
COBOL 289
función de tabla de OLE DB 352
Java 307
rutinas Java (DB2GENERAL) 315
VARCHAR FOR BIT DATA, tipo de datos
funciones definidas por el usuario en C/C++ 236

VARCHAR FOR BIT DATA, tipo de datos *(continuación)*
rutinas Java (DB2GENERAL) 315
VARGRAPHIC, tipo de datos
COBOL 289
función de tabla de OLE DB 352
funciones definidas por el usuario en C/C++ 236
Java 307
rutinas Java (DB2GENERAL) 315
variables
procedimientos de SQL 91, 98
variables del lenguaje principal
COBOL, tipos de datos 289
variables gráficas del lenguaje principal
rutinas 257
vinculación
procedimientos de SQL 111
rutinas 57, 356
Visual Explain
guía de aprendizaje 386

W

wchar_t, tipo de datos
rutinas C/C++ 236
WCHARTYPE NOCONVERT, opción del precompilador 257
Windows
rutinas SQLJ 334

X

XML
parámetros
compromisos y retrotracciones 95
procedimientos 92
tipo de datos 149
variables en procedimientos 92

SC11-3503-03

Spine information:

DB2 Versión 9.5 para Linux, UNIX y Windows **Versión 9 Release 5**

Desarrollo de rutinas definidas por el usuario (SQL y externas)