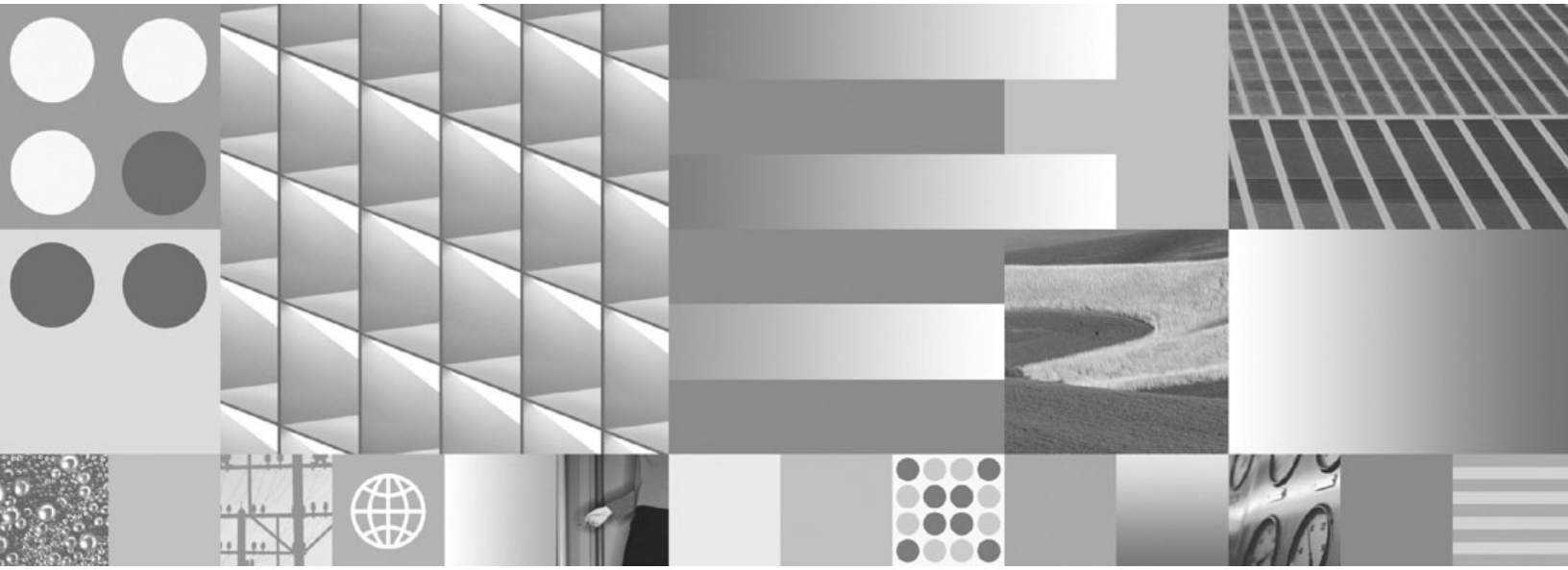
DB2 Versión 9.5
para Linux, UNIX y Windows

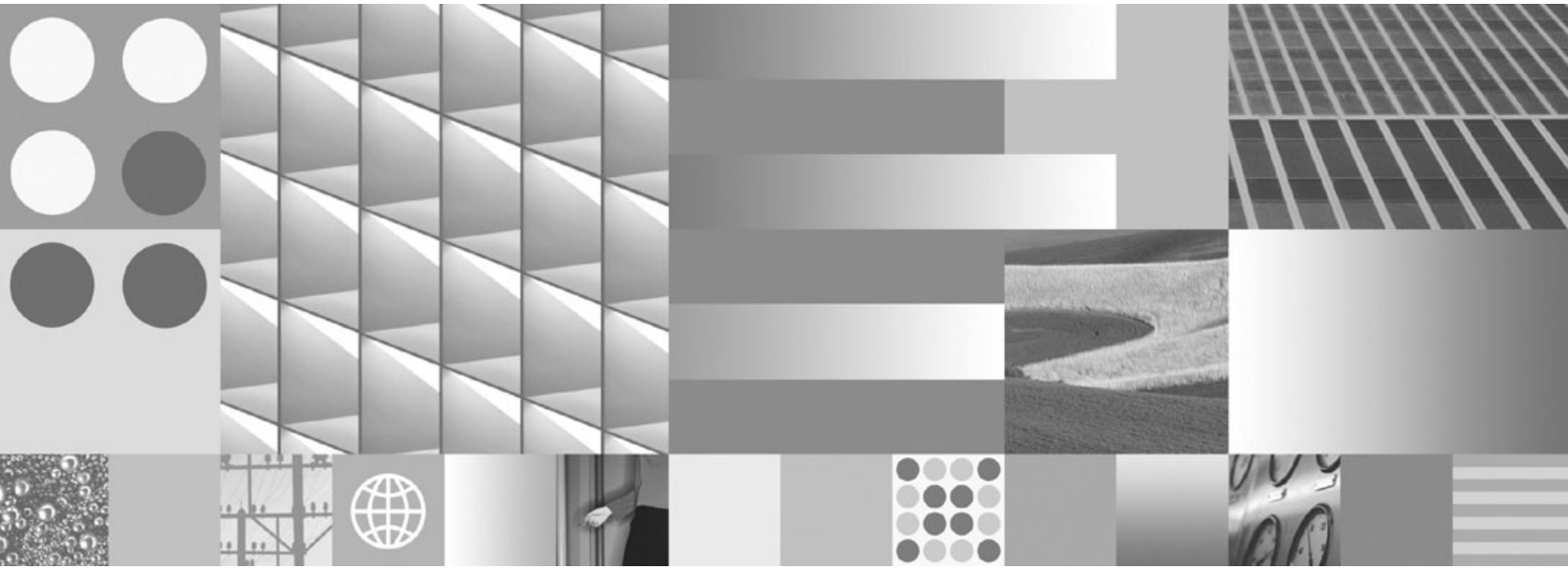


Desarrollo de aplicaciones ADO.NET y OLE DB

Actualizado en marzo de 2008

SC11-3499-01

DB2 Versión 9.5
para Linux, UNIX y Windows



Desarrollo de aplicaciones ADO.NET y OLE DB

Actualizado en marzo de 2008

SC11-3499-01

Nota

Antes de utilizar esta información y el producto al que da soporte, lea la información general contenida en el apartado Apéndice B, “Avisos”, en la página 177.

Nota de edición

Esta publicación es la traducción del original inglés: DB2 Version 9.5 for Linux, UNIX, and Windows - Developing ADO.NET and OLE DB Applications, (SC23-5851-01).

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

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

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

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

Contenido

Capítulo 1. Desarrollo de aplicaciones ADO.NET

En años recientes, Microsoft ha estado promocionando una nueva plataforma de desarrollo de software para Windows, conocida como .NET Framework. .NET Framework es la alternativa de Microsoft para la tecnología de Component Object Model (COM). Los puntos siguientes resaltan las características claves de .NET Framework:

- Puede codificar las aplicaciones .NET en más de cuarenta lenguajes de programación diferentes. Los lenguajes más populares para el desarrollo de .NET son C# y Visual Basic .NET.
- La biblioteca de clase de .NET Framework proporciona los bloques de creación con los que han de crearse las aplicaciones .NET. Esta biblioteca de clase tiene un lenguaje agnóstico y proporciona interfaces al sistema operativo y servicios de aplicación.
- La aplicación .NET (sin tener en cuenta el lenguaje) se compila en Intermediate Language (IL), un tipo de código de bytes.
- Common Language Runtime (CLR) es el corazón de .NET Framework, compilando el código de IL al vuelo y ejecutándolo a continuación. Al ejecutar el código IL compilado, CLR activa los objetos, verifica su autorización de seguridad, asigna su memoria, los ejecuta y limpia su memoria una vez haya finalizado la ejecución.

Por medio de estas características, .NET Framework facilita una amplia gama de implementaciones de aplicación (por ejemplo, los formatos de Windows, los formatos de Web y los servicios Web), el desarrollo de aplicación rápida y el desarrollo de aplicación protegido. COM y COM+ han demostrado que eran inadecuados o engorrosos para todas las características mencionadas anteriormente.

.NET Framework proporciona un amplio soporte de acceso de datos por medio de ADO.NET. ADO.NET da soporte tanto al acceso conectado como al desconectado. El componente clave del acceso de datos desconectado en ADO.NET es la clase DataSet, cuyas instancias actúan como antememoria de la base de datos que reside en la memoria de la aplicación.

Tanto para el acceso conectado como para el desconectado, las aplicaciones utilizan bases de datos a través de lo que se conoce como proveedor de datos. Diversos productos de la base de datos incluyen su propios proveedores de datos .NET, incluyendo DB2 para Windows.

Un proveedor de datos .NET presenta implementaciones de las siguientes clases básicas:

- Conexión: Establece y gestiona una conexión de base de datos.
- Mandato: Ejecuta una sentencia SQL en una base de datos.
- DataReader: Lee y devuelve datos del conjunto de resultados de una base de datos.
- DataAdapter: Enlaza una instancia de DataSet a una base de datos. Por medio de una instancia de DataAdapter, DataSet puede leer y grabar datos de tabla de la base de datos.

Microsoft proporciona dos proveedores de datos: OLE DB .NET Data Provider y ODBC .NET Data Provider. OLE DB .NET Data Provider es un proveedor puente que alimenta las peticiones de ADO.NET a IBM OLE DB Provider (por medio del módulo interop COM). ODBC .NET Data Provider es un proveedor puente que alimenta las peticiones ADO.NET a IBM ODBC Driver. Este proveedor de datos .NET no es recomendable par acceder a las bases de datos de la familia de DB2. IBM Data Server Provider para .NET es un proveedor de datos ADO.NET gestionado y de alto rendimiento. Es el proveedor de datos .NET recomendado para las bases de datos de la familia de DB2. El acceso a las bases de datos ADO.NET utilizando IBM Data Server Provider para .NET presenta menos restricciones y proporciona un rendimiento notablemente superior al de los proveedores puente OLE DB y ODBC .NET.

Despliegue de aplicaciones .NET (Windows)

Para simplificar el despliegue de aplicaciones, IBM proporciona IBM Data Server Driver para ODBC, CLI y .NET, un cliente pequeño ideal para distribuir aplicaciones de forma masiva en entornos de distribución. En su lugar, puede utilizar IBM Data Server Runtime Client, si desea las características adicionales de dicho cliente y no las de IBM Data Server Driver para ODBC, CLI y .NET.

Requisitos previos

- Antes del despliegue, debe crear la aplicación .NET, lo que puede llevar a cabo con Visual Studio o con la línea de mandatos. Para obtener más información acerca de cómo crear aplicaciones .NET, consulte las tareas relacionadas.
- Los sistemas que utilice para crear aplicaciones y sistemas .NET donde desplegará las aplicaciones .NET deben tener una versión soportada del sistema operativo Windows, además de otro software, como se describe en “Software de desarrollo .NET soportado” en la página 3:
  - Sistemas de creación
    - Sistema operativo Windows
    - Visual Studio
    - .NET Framework Redistributable Package
    - .NET Framework Software Development Kit
  - Sistemas de despliegue
    - Sistema operativo Windows
    - .NET Framework Redistributable Package

Para desplegar una aplicación .NET:

1. Instale IBM Data Server Driver para ODBC, CLI y .NET en los sistemas en los que desplegará la aplicación. Durante la instalación, establezca la instalación de IBM Data Server Driver para ODBC, CLI y .NET como la copia de la interfaz del cliente de base de datos por omisión.

   Nota: Cualquier aplicación de base de datos existente que se ejecuta en el servidor de datos IBM utilizará esta nueva instalación de IBM Data Server Driver para ODBC, CLI y .NET. Compruebe estas aplicaciones en el nuevo controlador antes de utilizar la aplicación .NET desplegada.
2. Instale la aplicación creada en los sistemas en los que se ejecutará la aplicación.

Software de desarrollo .NET soportado

Para desarrollar y desplegar aplicaciones .NET que se ejecutan en servidores de datos IBM, deberá utilizar software de desarrollo y sistemas operativos compatibles.

Sistemas operativos soportados para el desarrollo y despliegue de aplicaciones .NET Framework 1.1

- Windows 2000
- Windows XP (edición de 32 bits)
- Windows Server 2003 (edición de 32 bits)
- Windows Vista (ediciones de 32 bits)

Sistemas operativos soportados para el desarrollo y despliegue de aplicaciones .NET Framework 2.0 y 3.0

- Windows 2000, Service Pack 3
- Windows XP, Service Pack 2 (ediciones de 32 bits y 64 bits)
- Windows Server 2003 (ediciones de 32 bits y 64 bits)
- Windows Vista (ediciones de 32 bits y 64 bits)

Software de desarrollo soportado para aplicaciones .NET Framework

Además de un cliente DB2, necesitará una de las opciones siguientes para desarrollar aplicaciones .NET Framework.

- Visual Studio 2003 (para aplicaciones .NET Framework 1.1)
- Visual Studio 2005 (para aplicaciones .NET Framework 2.0 y 3.0)

Software de despliegue soportado para aplicaciones .NET Framework

Además de un cliente de ejecución DB2, necesitará una de las tres opciones para desplegar aplicaciones .NET Framework. En la mayoría de los casos, con una instalación de Windows se incluye una de estas opciones.

- .NET Framework Versión 1.1 Redistributable Package (para aplicaciones .NET Framework 1.1)
- .NET Framework Versión 2.0 Redistributable Package (para aplicaciones .NET Framework 2.0)
- .NET Framework Versión 3.0 Redistributable Package (para aplicaciones .NET Framework 3.0)

Integración de DB2 en Visual Studio

IBM Database Add-Ins para Visual Studio 2003 y 2005 son un conjunto de características que se integran de modo imperceptible en el entorno de desarrollo de Visual Studio para que pueda trabajar con servidores de DB2 y desarrollar objetos, funciones y procedimientos de DB2.

Los IBM Database Add-Ins para Visual Studio 2003 y 2005 están diseñados para presentar una interfaz sencilla para bases de datos DB2. Por ejemplo, en vez de utilizar SQL, la creación de objetos de base de datos puede efectuarse utilizando asistentes. Y para las situaciones en las que necesite escribir código de SQL, el editor SQL de DB2 integrado tiene las siguientes características:

- Texto SQL en color para una mejor legibilidad
- Integración con la función Microsoft Visual Studio IntelliSense, que proporciona la cumplimentación automática inteligente al escribir scripts de DB2

Con los IBM Database Add-Ins para Visual Studio, podrá:

- Abrir diversas herramientas de administración y desarrollo de DB2
- Crear y gestionar proyectos de DB2 en el Explorador de soluciones
- Acceder y gestionar conexiones de datos DB2 (en Visual Studio 2005 podrá hacer esto desde el Server Explorer; en Visual Studio 2003, podrá hacer esto desde el IBM Explorer)
- Crear y modificar scripts de DB2, incluyendo scripts para crear procedimientos almacenados, funciones, tablas, vistas, índices y activadores.

A continuación se proporcionan los medios por los que los IBM Database Add-Ins para Visual Studio pueden instalarse en el sistema.

Visual Studio 2003

Los IBM Database Add-Ins para Visual Studio 2003 se incluyen con DB2 Client y con los servidores de DB2. La instalación de DB2 detecta la presencia de Visual Studio 2003 y si está instalado, se registran los add-ins. Si instala Visual Studio 2003 después de instalar un producto DB2, ejecute el programa de utilidad "Registrar módulos adicionales de Visual Studio" en el menú de inicio de la instancia de DB2.

Visual Studio 2005

Los IBM Database Add-Ins para Visual Studio 2005 se incluyen como componente de instalación independiente con DB2 Client y con los servidores DB2. Una vez se haya acabado de instalar el producto de DB2, aparecerá una opción para instalar los IBM Database Add-Ins para Visual Studio 2005. Si no ha instalado Visual Studio 2005 en el sistema, no se instalarán los add-ins. Una vez haya instalado Visual Studio 2005, podrá instalar los add-ins en cualquier momento desde el menú de configuración del producto de DB2.

Capítulo 2. 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.

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” en la página 7
- “Creación de rutinas externas” en la página 24
- “Gestión de bibliotecas y clases de rutinas externas” en la página 28
- “Interfaces API y lenguajes de programación soportados para el desarrollo de rutinas externas” en la página 18
- “Soporte de 32 bits y 64 bits para rutinas externas” en la página 32
- “Estilos de parámetros de rutinas externas” en la página 26
- “Restricciones para rutinas externas” en la página 34

Cuando haya comprendido los conceptos sobre rutinas externas, puede consultar:

- “Creación de rutinas externas” en la página 38

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.

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 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.

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" en la página 8
- "Modelo de proceso de los métodos y las funciones escalares externas" en la página 9

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

- 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.

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 1. 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 1. 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 1. 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 1. 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.	• Las rutinas CLR solo se pueden crear con la cláusula FENCED NOT THREADSAFE para minimizar la posibilidad de una interrupción del gestor de bases de datos durante el tiempo de ejecución. Esto puede tener un cierto efecto negativo sobre el rendimiento.	• Las rutinas CLR solo se pueden crear con la cláusula FENCED NOT THREADSAFE. Por lo tanto, son seguras porque se ejecutarán fuera del gestor de bases de datos en un proceso db2fmp aparte.	• No hay información disponible.	• Consulte el tema ″Restricciones de las rutinas CLR .NET″.

Tabla 1. 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 1. 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.

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.

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 2. 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 2. 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 2. 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.

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.

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 3. 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)

Tabla 3. Soporte para rutinas externas de 32 y de 64 bits (continuación)

Ancho de bits de la rutina	Servidor de 32 bits	Servidor de 64 bits
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 de 64 bits. 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 de 64 bits. 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.

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.

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 como consecuencia 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.

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.

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:
  - Capítulo 2, “Rutinas externas”, en la página 5

– “Creación de rutinas externas” en la página 24

- 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 34

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 26
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

Capítulo 3. Rutinas CLR (common language runtime) de .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 5
- “Representación de tipo de datos de SQL en rutinas CLR de .NET” en la página 43
- “Parámetros de rutinas .NET CLR” en la página 45
- “Devolución de conjuntos de resultados desde procedimientos .NET CLR” en la página 48
- “Restricciones de las rutinas CLR .NET” en la página 50
- “Errores relacionados con rutinas CLR .NET” en la página 61

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 52
- “Ejemplos de procedimientos CLR de .NET en C#” en la página 64
- “Ejemplos de funciones CLR de .NET en C#” en la página 99

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.

Sistemas operativos soportados para el desarrollo de rutinas .NET CLR con .NET Framework Versiones 1.1, 2.0 o 3.0

- Windows 2000
- Windows XP (edición de 32 bits)
- Windows Server 2003 (edición de 32 bits)

Servidores y clientes de bases de datos de DB2 para el desarrollo de rutinas .NET CLR

Se deben instalar los siguientes servidores y clientes mínimos de DB2:

- Servidor DB2: la versión mínima soportada es DB2 versión 8.2.
- Cliente DB2: la versión mínima soportada es DB2 versión 7.2.

Software de desarrollo necesario para rutinas .NET CLR

Uno de los dos productos de software siguientes debe estar instalado en el mismo sistema que el servidor de bases de datos de DB2:

- Microsoft .NET Framework, versión 1.1
- Microsoft .NET Framework, versión 2.0

Microsoft .NET Framework está disponible de forma independiente o como parte de uno de los siguientes Kits de desarrollo de software:

- Kit de desarrollo de software de Microsoft .NET Framework versión 1.1
- Kit de desarrollo de software de Microsoft .NET Framework versión 2.0
- Kit de desarrollo de software de Microsoft .NET Framework versión 3.0

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#.

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 DB2 Development Add-In para Microsoft Visual Studio .NET 1.2

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 de DB2Type**	**Tipo de datos de DB2**	**Tipo de datos de 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

1. Estos tipos de datos no se soportan 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 son aplicables 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.

Categoría	Clases y estructuras de DB2Types	Tipo de datos de DB2Type	Tipo de datos de DB2	Tipo de datos de Informix	Tipo de datos .NET
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
Binary data	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[]

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:

- Devolución de conjuntos de resultados desde procedimientos CLR

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 52

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.

Tipos de datos no soportados 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 GRAPHIC
- ROWID

Ejecución de una rutina CLR de 32 bits en una instancia de 64 bits

Las rutinas CLR no se pueden ejecutar en instancias de 64 bits, porque .NET Framework no se puede instalar en sistemas operativos de 64 bits en este momento.

CLR .NET no soportado para la implementación de plugins de seguridad

CLR .NET no está soportado para compilar y enlazar el código fuente de 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 DB2 Database Development Add-In para Visual Studio .NET 1.2
- Utilizar la ventana de mandatos de DB2

En general resulta más fácil crear rutinas .NET CLR utilizando DB2 Database Development Add-In para Visual Studio .NET 1.2. 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 las Capítulo 3, "Rutinas CLR (common language runtime) de .NET", en la página 41.
- 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 50

Cree rutinas .NET CLR desde una de las interfaces siguientes:

- Visual Studio .NET cuando también esté instalado IBM DB2 Development Add-In para Microsoft Visual Studio .NET 1.2. 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:
  - Capítulo 3, "Rutinas CLR (common language runtime) de .NET", en la página 41
- El servidor de bases de datos debe ejecutar un sistema operativo Windows que dé soporte a Microsoft .NET Framework.
- El producto .NET Framework, versión 1.1 o 2.0, debe estar instalado en el servidor. .NET Framework está disponible de forma independiente o forma parte del Kit de desarrollo de software de Microsoft .NET Framework 1.1 o del Kit de desarrollo de software .NET Framework 2.0.
- 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 50

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 43
   - 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 45
   - 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 48

- 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. Si desea obtener información sobre la forma en que construir rutinas CLR .NET que accedan a DB2, consulte el siguiente tema:
   - "Creación de rutinas CLR (Common Language Runtime) .NET" en *Desarrollo de aplicaciones ADO.NET y OLE DB*
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 DB2 Database Development Add-In para Visual Studio .NET 1.2
- 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 va 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 va 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 va 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 va 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 tres versiones de la infraestructura .NET a las que se da soporte para las aplicaciones: versión 1.1, versión 2.0 y versión 3.0. Existe una biblioteca de enlaces dinámicos para cada una de ellas en subdirectorios independientes. Para .NET Framework versión 1.1, utilice el subdirectorio. `"%DB2PATH%"\bin\netf11`. Para .NET Framework versión 2.0 y 3.0, 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 tres versiones de la infraestructura .NET a las que se da soporte para las aplicaciones: versión 1.1, versión 2.0 y versión 3.0. Existe una biblioteca de enlaces dinámicos para cada una de ellas en subdirectorios independientes. Para .NET Framework versión 1.1, utilice el subdirectorio. `"%DB2PATH%"\bin\netf11`. Para .NET Framework versión 2.0 y 3.0, 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 49

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 99

Antes de trabajar con los ejemplos de procedimientos CLR, puede ser conveniente que lea los temas sobre los conceptos siguientes:

- Capítulo 3, "Rutinas CLR (common language runtime) de .NET", en la página 41
- "Creación de rutinas CLR .NET desde la ventana de mandatos de DB2" en la página 52
- "Creación de rutinas CLR (Common Language Runtime) .NET" en *Desarrollo de aplicaciones ADO.NET y OLE DB*

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 4. 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 5. 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 6. 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 6. 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 7. Código para crear un procedimiento en C# del estilo de parámetro 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 7. Código para crear un procedimiento en C# del estilo de parámetro 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 8. 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 9. 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 10. 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 10. 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 80

Antes de trabajar con los ejemplos de UDF CLR, puede ser conveniente que lea los temas sobre los conceptos siguientes:

- Capítulo 3, "Rutinas CLR (common language runtime) de .NET", en la página 41
- "Creación de rutinas CLR .NET desde la ventana de mandatos de DB2" en la página 52
- "Funciones escalares externas" en la página 8
- "Creación de rutinas CLR (Common Language Runtime) .NET" en *Desarrollo de aplicaciones ADO.NET y OLE DB*

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 11. 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 12. 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 12. 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 12. 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 13. 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 13. 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 75

Antes de trabajar con los ejemplos de procedimientos CLR, puede ser conveniente que lea los temas sobre los conceptos siguientes:

- Capítulo 3, "Rutinas CLR (common language runtime) de .NET", en la página 41
- "Creación de rutinas CLR .NET desde la ventana de mandatos de DB2" en la página 52
- "Beneficios del uso de rutinas" en la página 5
- "Creación de rutinas CLR (Common Language Runtime) .NET" en *Desarrollo de aplicaciones ADO.NET y OLE DB*

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 14. 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 15. 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 15. 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 16. 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 17. 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 17. 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 18. 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 19. 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 20. 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 20. 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
- "Creación de rutinas CLR (Common Language Runtime) .NET" en *Desarrollo de aplicaciones ADO.NET y OLE DB*

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 21. 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 22. 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 22. 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 23. 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 24. 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 24. 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 64

Antes de trabajar con los ejemplos de UDF CLR, puede ser conveniente que lea los temas sobre los conceptos siguientes:

• Capítulo 3, "Rutinas CLR (common language runtime) de .NET", en la página 41
• "Creación de rutinas CLR .NET desde la ventana de mandatos de DB2" en la página 52
• "Funciones escalares externas" en la página 8
• "Creación de rutinas CLR (Common Language Runtime) .NET" en *Desarrollo de aplicaciones ADO.NET y OLE DB*

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 25. 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 26. 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 26. 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 26. 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 27. 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 27. 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;
      }
   }
}
```

Capítulo 4. IBM Data Server Provider para .NET

IBM Data Server Provider para .NET amplía el soporte de DB2 para la interfaz ADO.NET. IBM Data Server Provider para .NET ofrece un acceso seguro y de alto rendimiento a servidores de datos IBM.

IBM Data Server Provider para .NET permite que sus aplicaciones .NET accedan a los siguientes sistemas de gestión de bases de datos:

- DB2 versión 9 (o posterior) para Linux, UNIX y Windows
- DB2 Universal Database versión 8 para sistemas Windows, UNIX y Linux
- DB2 Universal Database versión 6 (o posterior) para OS/390 y z/OS, a través de DB2 Connect
- DB2 Universal Database versión 5, Release 1 (o posterior) para AS/400 e iSeries, a través de DB2 Connect
- DB2 Universal Database versión 7.3 (o posterior) para VSE y VM, a través de DB2 Connect
- IBM Informix Dynamic Server, versión 11.10 o posterior
- IBM UniData, versión 7.1.11 o posterior
- IBM UniVerse, versión 10.2 o posterior

Para desarrollar y ejecutar aplicaciones que utilizan Data Server Provider para .NET necesita .NET Framework, versión 1.1, 2.0 o 3.0.

Además de IBM Data Server Provider para .NET, IBM Database Development Add-Ins le permite desarrollar de forma rápida y sencilla aplicaciones .NET para servidores de datos IBM utilizando Visual Studio 2005. También puede utilizar los add-ins para crear objetos de base de datos como índices y tablas, y desarrollar objetos del servidor, como procedimientos almacenados de SQL y funciones definidas por el usuario.

Requisitos del sistema de bases de datos IBM Data Server Provider para .NET

IBM Data Server Provider para .NET permite que las aplicaciones .NET accedan a los siguientes sistemas de gestión de bases de datos:

- DB2 versión 9 (o posterior) para Linux, UNIX y Windows
- DB2 Universal Database versión 8 para Linux, UNIX y Windows
- DB2 Universal Database versión 6 (o posterior) para OS/390 y z/OS, a través de DB2 Connect
- DB2 Universal Database versión 5, Release 1 (o posterior) para AS/400 e iSeries, a través de DB2 Connect
- DB2 Universal Database versión 7.3 (o posterior) para VSE y VM, a través de DB2 Connect
- IBM Informix Dynamic Server, versión 11.10 o posterior
- IBM UniData, versión 7.1.11 o posterior
- IBM UniVerse, versión 10.2 o posterior

Antes de utilizar el programa de instalación cliente o servidor DB2 para instalar IBM Data Provider para .NET, debe tener .NET Framework (versión 1.1, 2.0 o 3.0)

instalado en el sistema. Si .NET Framework no está instalado, el programa de instalación de cliente o servidor de DB2 no instalará IBM Data Server Provider para .NET.

Para DB2 Universal Database para AS/400 e iSeries, es necesario el siguiente arreglo de programa en el servidor: APAR ii13348.

Soporte de 32 bits y 64 bits para aplicaciones ADO.NET

Los proveedores de datos de IBM para .NET dan soporte a las aplicaciones .NET de 32 y 64 bits.

A continuación se ofrece una lista de los proveedores de datos .NET que se suministran con los clientes y servidores DB2 versión 9 (o superior) y los niveles de soporte de 32 bits y 64 bits correspondientes.

Tabla 28. Soporte de aplicaciones de 32 y 64 bits en los proveedores de datos de IBM para .NET

Proveedor de datos .NET	soporte de 32 bits	soporte de 64 bits
IBM Data Server Provider para .NET, Framework versión 1.1	Sí	No
IBM Data Server Provider para .NET, Framework versión 2.0	Sí	Sí
IBM Data Server Provider para .NET, Framework versión 3.0	Sí	Sí

Nota: Los procedimientos almacenados y las funciones definidas por el usuario CLR sólo reciben soporte en las ediciones de 32 bits de IBM Data Server Provider para .NET.

IBM Data Server Provider para .NET, Framework versión 2.0

Existen ediciones de 32 bits y 64 bits de IBM Data Server Provider para .NET que dan soporte a las ediciones de 32 bits y 64 bits de CLR de .NET Framework versión 2.0 respectivamente. Durante la instalación del software de cliente o servidor DB2 o DB2 Connect también se instalará una de estas dos ediciones de IBM Data Server Provider para .NET:

Para sistemas Windows en AMD e Intel (x86) de 32 bits
La edición de 32 bits de IBM Data Server Provider para .NET, Framework 2.0, y también para 1.1 y 3.0, se instalan con DB2 versión 9 (o posterior) o DB2 Connect.

Para sistemas Windows en AMD64 e Intel EM64T (x64) (fixpack 2 y posteriores)
Sólo las ediciones de 64 bits de IBM Data Server Provider para .NET se instalan con DB2 versión 9 (o posterior) o DB2 Connect. IBM Data Server Provider para .NET, Framework 1.1 no se instala. Las ediciones de 64 bits de IBM Data Server Provider para .NET no dan soporte a la arquitectura IA-64.

Puede ejecutar aplicaciones .NET de 32 bits en una instancia de Windows de 64 bits, utilizando WOW64, pero también necesitará una edición de 32 bits de IBM Data Server Provider para .NET. Para obtener IBM Data Server Provider para .NET de 32 bits, puede instalar DB2 Connect o DB2 versión 9 para Windows en sistemas AMD y Intel, cliente o servidor, de 32 bits en su sistema de 64 bits.

Aplicaciones de programación para utilizar IBM Data Server Provider para .NET

Codificación genérica con las clases básicas comunes de ADO.NET

.NET Framework, versiones 2.0 y 3.0, ofrece un espacio de nombres denominado `System.Data.Common`, que presenta un conjunto de clases básicas que puede compartirse con cualquier proveedor de datos .NET. Esta acción facilita un enfoque de desarrollo de la aplicación de base de datos ADO.NET genérico, que resalta una interfaz de programación constante.

Las clases principales de IBM Data Server Provider para .NET, Framework 2.0 y 3.0, se heredan de las clases básicas `System.Data.Common`. En consecuencia, las aplicaciones ADO.NET genéricas funcionarán con las bases de datos DB2 y otras bases de datos soportadas a través de IBM Data Server Provider para .NET.

El siguiente C# demuestra un enfoque genérico para establecer una conexión de base de datos.

```
DbProviderFactory factory = DbProviderFactories.GetFactory("IBM.Data.DB2");
DbConnection conn = factory.CreateConnection();
DbConnectionStringBuilder sb = factory.CreateConnectionStringBuilder();

if( sb.ContainsKey( "Database" ) )
{
   sb.Remove( "database" );
   sb.Add( "database", "SAMPLE" );
}

conn.ConnectionString = sb.ConnectionString;

conn.Open();
```

El objeto `DbProviderFactory` es el punto en el que comienza cualquier aplicación ADO.NET genérica. Este objeto crea instancias genéricas de objetos de proveedor de datos .NET, como por ejemplo conexiones, adaptadores de datos, mandatos y lectores de datos, que funcionan con un producto de base de datos específico. En el caso del ejemplo anterior, la serie `"IBM.Data.DB2"` pasada al método `GetFactory` identifica de forma unívoca IBM Data Server Provider para .NET, y da lugar a la inicialización de una instancia de `DbProviderFactory` que crea instancias de objeto de proveedor de bases de datos específicas de IBM Data Server Provider para .NET. El objeto `DbConnection` puede conectarse a las bases de datos de la familia de DB2, igual que el objeto `DB2Connection`, que se hereda realmente desde `DbConnection`. Utilizando la clase `DbConnectionStringBuilder`, podrá determinar las palabras clave de la serie de conexión para un proveedor de datos y generar una serie de conexión personalizada. El código del ejemplo anterior comprueba si en IBM Data Server Provider para .NET existe una palabra clave denominada `"database"` y, en caso afirmativo, genera una serie de conexión para conectar con la base de datos SAMPLE.

Conexión a una base de datos desde una aplicación utilizando IBM Data Server Provider para .NET

Cuando se utiliza IBM Data Server Provider para .NET, se establece una conexión de base de datos a través de la clase `DB2Connection`. Primero, el usuario debe crear una serie de caracteres para contener los parámetros de conexión.

Son ejemplos de series de conexión:

```
String connectString = "Database=SAMPLE";
// Cuando se utiliza, intenta conectar con la base de datos SAMPLE.
String cs = "Server=srv:50000;Database=SAMPLE;UID=db2adm;PWD=ab1d;Connect Timeout=30";
// Cuando se utiliza, intenta conectar con la base de datos SAMPLE del servidor
// 'srv' a través del puerto 50000 y utilizando 'db2adm' y 'ab1d' como
// ID de usuario y contraseña, respectivamente. Si el intento de conexión
// tarda más de treinta segundos, el intento terminará y se generará un error.
```

Para crear la conexión de base de datos, pase la serie de conexión `connectString` al constructor de `DB2Connection`. Luego utilice el método `Open` del objeto `DB2Connection` para conectar formalmente con la base de datos especificada en `connectString`.

- Conexión a una base de datos en C#:

```
String connectString = "Database=SAMPLE";
DB2Connection conn = new DB2Connection(connectString);
conn.Open();
return conn;
```

- Conexión a una base de datos en Visual Basic .NET:

```
Dim connectString As String = "Database=SAMPLE"
Dim conn As DB2Connection = new DB2Connection(connectString)
conn.Open()
Return conn
```

Agrupación de conexiones con IBM Data Server Provider para .NET

La primera vez que se abra una conexión en la base de datos de DB2, se creará una agrupación de conexiones. Cuando se cierren las conexiones, éstas entran la agrupación, preparada para que la utilicen otras aplicaciones que necesitan conexiones. IBM Data Server Provider para .NET permite por omisión la agrupación de conexiones. Puede desactivar la agrupación de conexiones utilizando el par de palabra clave/valor de la serie de conexiones `Pooling=false`.

Puede controlar el comportamiento de la agrupación de conexiones estableciendo las palabras clave de la serie de conexiones para lo siguiente:

- El tamaño de agrupación mínimo y máximo (tamaño de agrupación mín, tamaño de agrupación máx)
- El tiempo que una conexión puede estar desocupada antes de devolverla a la agrupación (tiempo de vida para la conexión)
- Si la conexión actual se colocará o no en la agrupación de conexiones cuando se cierre (restauración de conexión)

Creación de una conexión acreditada mediante IBM Data Server Provider para .NET

Empezando con la versión 9.5 Fix Pack 1, las aplicaciones .NET soportan contextos acreditados utilizando palabras clave de series de contexto.

Las siguientes palabras clave están disponibles en la serie de conexión:

- TrustedContextSystemUserID o tcsuid, que especifica el contexto acreditado SYSTEM AUTHID que debe utilizarse con la conexión.
- TrustedContextSystemPassword o tcspwd, que especifica la contraseña correspondiente al contexto acreditado SYSYTEM AUTHID que debe utilizarse con la conexión.

Si se especifica la palabra clave TrustedContextSystemPassword sin un valor de palabra clave TrustedContextSystemUserID, se lanza una excepción InvalidArgument. La palabra clave UserID también es necesaria en un escenario de contexto acreditado.

Ejemplo

Supongamos que se ha establecido un contexto acreditado en un servidor con la siguiente información:

```
CREATE TRUSTED CONTEXT ctxName1
BASED UPON CONNECTION USING SYSTEM AUTHID masteruser
ATTRIBUTES ( PROTOCOL 'TCPIP',
             ADDRESS '9.26.146.201',
             ENCRYPTION 'NONE' )
ENABLE
WITH USE FOR userapp1 WITH AUTHENTICATION, userapp2 WITH AUTHENTICATION;
```

El SYSTEM AUTHID, masteruser (usuario maestro), tiene una contraseña correspondiente. Cada usuario/aplicación específica, userapp1 y userapp2, tiene una contraseña correspondiente, passapp1 y passapp2.

Para utilizar este contexto acreditado, las aplicaciones emitirán series de conexión de la siguiente manera:

- Aplicación 1

  ```
  database=db;server=foobar:446;UserID=userapp1;Password=passapp1;
  TrustedContextSystemUserID=masteruser;TrustedContextSystemPassword=masterpassword
  ```

- Aplicación 2

  ```
  database=db;server=foobar:446;UserID=userapp2;Password=passapp2;
  TrustedContextSystemUserID=masteruser;TrustedContextSystemPassword=masterpassword
  ```

Nota: La palabra clave UserID corresponde al usuario final de la conexión en una situación de contexto acreditado, como en las aplicaciones estándar.

De este modo, un simple programa .NET tendría la siguiente apariencia:

```
DB2Connection conn = new DB2Connection();

conn.ConnectionString = "database=db;server=foobar:446;UserID=userapp1;Password=passapp1;
TrustedContextSystemUserID=masteruser;TrustedContextSystemPassword=masterpassword;"

conn.Open();

// Procesar como userapp1, como tablas de consulta

conn.Close();

conn.ConnectionString = "database=db;server=foobar:446;UserID=userapp2;Password=passapp2;
TrustedContextSystemUserID=masteruser;TrustedContextSystemPassword=masterpassword;"

conn.Open();

// Procesar como userapp2

conn.Close();
```

Si el proceso de contexto acreditado falla porque no se ha configurado ningún contexto acreditado en el servidor o dicho servidor no soporta contextos acreditados, se lanzará un error con SQLCODE CLI0197E. Si el valor de la palabra clave TrustedContextSystemUserID no es válido (por ejemplo, es demasiado largo), se lanzará un error con SQLCODE CLI0124E. Es posible que el servidor informe de

un error con SQLCODE SQL1046N, SQL30082N o SQL0969N con un código de error nativo de -20361. Ninguno de estos errores puede hacer que falle Open().

Nota: El proceso de contexto acreditado se producirá en la siguiente comunicación con el servidor.

Representación de tipo de datos de SQL en aplicaciones de de base de datos ADO.NET

Las aplicaciones de base de datos de ADO.NET pueden hacer referencia a valores de tipos de datos de SQL de DB2 como valores de parámetros que deban utilizarse como parte de la ejecución de la sentencia de SQL y como variables; no obstante, debe utilizar valores de tipo de datos de IBM Data Server Provider para .NET y valores de tipos de datos de .NET Framework para garantizar que no se produzca ningún truncamiento o pérdida de datos al acceder o recuperar los valores.

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 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.

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 de DB2Type	Tipo de datos de DB2	Tipo de datos de 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)[58]		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[6]	XML		Byte[]
Character data	DB2String	Char	CHAR	CHAR	String
	DB2String	VarChar	VARCHAR	VARCHAR	String
	DB2String	LongVarChar[5]	LONG VARCHAR	LVARCHAR	String
Binary data	DB2Binary	Binary	CHAR FOR BIT DATA		Byte[]
	DB2Binary	Binary[7]	BINARY		Byte[]
	DB2Binary	VarBinary[7]	VARBINARY		Byte[]
	DB2Binary	LongVarBinary[5]	LONG VARCHAR FOR BIT DATA		Byte[]

5. Estos tipos de datos no se soportan como parámetros en rutinas de tiempo de ejecución de lenguaje común de DB2 .NET.
6. Una propiedad DB2ParameterClass.ParameterName del tipo DB2Type.Xml puede aceptar variables de los siguientes tipos: String, byte[], DB2Xml y XmlReader.
7. Estos tipos de datos sólo son aplicables a DB2 UDB para z/OS.
8. 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.

Categoría	Clases y estructuras de DB2Types	Tipo de datos de DB2Type	Tipo de datos de DB2	Tipo de datos de Informix	Tipo de datos .NET
Graphic data	DB2String	Graphic	GRAPHIC		String
	DB2String	VarGraphic	VARGRAPHIC		String
	DB2String	LongVarGraphic[5]	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[]

Ejecución de sentencias de SQL desde una aplicación utilizando IBM Data Server Provider para .NET

Cuando se utiliza IBM Data Server Provider para .NET, la ejecución de sentencias de SQL se realiza a través de una clase `DB2Command` utilizando sus métodos `ExecuteReader()` y `ExecuteNonQuery()`, y sus propiedades `CommandText`, `CommandType` y `Transaction`. Para las sentencias de SQL que generan datos de salida, se debe utilizar el método `ExecuteReader()` y sus resultados se pueden recuperar de un objeto `DB2DataReader`. Para todas las demás sentencias de SQL, se debe utilizar el método `ExecuteNonQuery()`. La propiedad `Transaction` del objeto `DB2Command` se debe inicializar para un objeto `DB2Transaction`. El objeto `DB2Transaction` es el encargado de retrotraer y confirmar las transacciones de base de datos.

Ejecución de una sentencia UPDATE en C#:

```
// se supone la existencia de una conexión DB2Connection
DB2Command cmd = conn.CreateCommand();
DB2Transaction trans = conn.BeginTransaction();
cmd.Transaction = trans;
cmd.CommandText = "UPDATE staff " +
                  "  SET salary = (SELECT MIN(salary) " +
                  "  FROM staff " +
    "                   WHERE id >= 310) " +
                  "  WHERE id = 310";
cmd.ExecuteNonQuery();
```

Ejecución de una sentencia UPDATE en Visual Basic .NET:

```
' se supone la existencia de una conexión DB2Connection
DB2Command cmd = conn.CreateCommand();
DB2Transaction trans = conn.BeginTransaction();
cmd.Transaction = trans;
cmd.CommandText = "UPDATE staff " +
                  "  SET salary = (SELECT MIN(salary) " +
                  "  FROM staff " +
                  "                   WHERE id >= 310) " +
                  "  WHERE id = 310";
cmd.ExecuteNonQuery();
```

Ejecución de una sentencia SELECT en C#:

```
// se supone la existencia de una conexión DB2Connection
DB2Command cmd = conn.CreateCommand();
DB2Transaction trans = conn.BeginTransaction();
cmd.Transaction = trans;
```

```
cmd.CommandText = "SELECT deptnumb, location " +
                  "  FROM org " +
                  "  WHERE deptnumb < 25";
DB2DataReader reader = cmd.ExecuteReader();
```

Ejecución de una sentencia SELECT en Visual Basic .NET:

```
' se supone la existencia de una conexión DB2Connection
Dim cmd As DB2Command = conn.CreateCommand()
Dim trans As DB2Transaction = conn.BeginTransaction()
cmd.Transaction = trans
cmd.CommandText = "UPDATE staff " +
                  "  SET salary = (SELECT MIN(salary) " +
                  "  FROM staff " +
                  "                  WHERE id >= 310) " +
                  "  WHERE id = 310"
cmd.ExecuteNonQuery()
```

Una vez que su aplicación ha efectuado una transacción de base de datos, debe retrotraerla o confirmarla. Esto se realiza mediante los métodos `Commit()` y `Rollback()` de un objeto `DB2Transaction`.

Retrotracción o confirmación de una transacción en C#:

```
// se supone la utilización del objeto DB2Transaction conn
trans.Rollback();
...
trans.Commit();
```

Retrotracción o confirmación de una transacción en Visual Basic.NET:

```
' se supone la utilización del objeto DB2Transaction conn
trans.Rollback();
...
trans.Commit()
```

Lectura de conjuntos de resultados de una aplicación mediante IBM Data Server Provider for .NET

Cuando se utiliza IBM Data Server Provider para .NET, la lectura de los conjuntos de resultados se lleva a cabo a través del objeto `DB2DataReader`. Se utiliza el método `Read()` de `DB2DataReader` para avanzar hacia la fila siguiente del conjunto de resultados. Para extraer los datos de las columnas del resultado se utilizan los métodos `GetString()`, `GetInt32()`, `GetDecimal()`, y otros métodos existentes para todos los tipos de datos disponibles. El método `Close()` de `DB2DataReader` se utiliza para cerrar el objeto `DB2DataReader`, lo cual debe hacerse siempre al terminar de leer el resultado.

Lectura de un conjunto de resultados en C#:

```
// se supone la utilización de un lector DB2DataReader
Int16 deptnum = 0;
String location="";

// Visualizar los resultados de la consulta
while(reader.Read())
{
  deptnum = reader.GetInt16(0);
  location = reader.GetString(1);
  Console.WriteLine("    " + deptnum + " " + location);
}
reader.Close();
```

Lectura de un conjunto de resultados en Visual Basic .NET:

```
' se supone la utilización de un lector DB2DataReader
Dim deptnum As Int16 = 0
Dim location As String ""

' Visualizar los resultados de la consulta
Do While (reader.Read())
  deptnum = reader.GetInt16(0)
  location = reader.GetString(1)
  Console.WriteLine("    " &deptnum &" " &location)
Loop
reader.Close();
```

Invocación de procedimientos almacenados de una aplicación utilizando IBM Data Server Provider para .NET

Cuando utiliza IBM Data Server Provider para .NET, puede invocar procedimientos almacenados mediante un objeto DB2Command. El valor por omisión de la propiedad CommandType es CommandType.Text. Este valor es el apropiado para sentencias de SQL y también se puede utilizar para invocar procedimientos almacenados. Pero la invocación de procedimientos almacenados es más fácil si define el valor de CommandType como CommandType.StoredProcedure. En este caso, solo es necesario que especifique el nombre del procedimiento almacenado y los parámetros.

Los ejemplos siguientes muestran cómo invocar un procedimiento almacenado llamado INOUT_PARAM, con la propiedad CommandType establecida en CommandType.StoredProcedure o CommandType.Text.

Invocación de un procedimiento almacenado utilizando CommandType.StoredProcedure como valor de la propiedad CommandType de DB2Command en C#:

```
// se supone la existencia de una conexión DB2Connection
DB2Transaction trans = conn.BeginTransaction();
DB2Command cmd = conn.CreateCommand();
String procName = "INOUT_PARAM";
cmd.Transaction = trans;
cmd.CommandType = CommandType.StoredProcedure;
cmd.CommandText = procName;

// Registro de los parámetros de entrada-salida y parámetros de
// salida para DB2Command
...

// Invocar el procedimiento almacenado
Console.WriteLine("  Call stored procedure named " + procName);
cmd.ExecuteNonQuery();
```

Invocación de un procedimiento almacenado utilizando CommandType.Text como valor de la propiedad CommandType de DB2Command en C#:

```
// se supone la existencia de una conexión DB2Connection
DB2Transaction trans = conn.BeginTransaction();
DB2Command cmd = conn.CreateCommand();
String procName = "INOUT_PARAM";
String procCall = "CALL INOUT_PARAM (?, ?, ?)";
cmd.Transaction = trans;
cmd.CommandType = CommandType.Text;
cmd.CommandText = procCall;

// Registro de los parámetros de entrada-salida y parámetros de
// salida para DB2Command
...
```

```
// Invocar el procedimiento almacenado
Console.WriteLine("  Call stored procedure named " + procName);
cmd.ExecuteNonQuery();
```

Invocación de un procedimiento almacenado utilizando `CommandType.StoredProcedure` como valor de la propiedad CommandType de `DB2Command` en Visual Basic .NET:

```
' se supone la existencia de una conexión DB2Connection
Dim trans As DB2Transaction = conn.BeginTransaction()
Dim cmd As DB2Command = conn.CreateCommand()
Dim procName As String = "INOUT_PARAM"
cmd.Transaction = trans
cmd.CommandType = CommandType.StoredProcedure
cmd.CommandText = procName

' Registro de los parámetros de entrada-salida y parámetros de salida
' para DB2Command
...

' Invocación del procedimiento almacenado
Console.WriteLine("  Call stored procedure named " &procName)
cmd.ExecuteNonQuery()
```

Invocación de un procedimiento almacenado utilizando `CommandType.Text` como valor de la propiedad CommandType de `DB2Command` en Visual Basic .NET:

```
' se supone la existencia de una conexión DB2Connection
Dim trans As DB2Transaction = conn.BeginTransaction()
Dim cmd As DB2Command = conn.CreateCommand()
Dim procName As String = "INOUT_PARAM"
Dim procCall As String = "CALL INOUT_PARAM (?, ?, ?)"
cmd.Transaction = trans
cmd.CommandType = CommandType.Text
cmd.CommandText = procCall

' Registro de los parámetros de entrada-salida y parámetros de salida
' para DB2Command
...

' Invocación del procedimiento almacenado
Console.WriteLine("  Call stored procedure named " &procName)
cmd.ExecuteNonQuery()
```

Creación de aplicaciones .NET

Creación de aplicaciones Visual Basic .NET

DB2 proporciona un archivo de proceso por lotes, bldapp.bat, para compilar y enlazar las aplicaciones Visual Basic .NET de DB2, ubicadas en el directorio sqllib\samples\.NET\vb junto a los programas de ejemplo que pueden crearse con este archivo. El archivo de proceso por lotes toma un parámetro, %1, para el nombre del archivo fuente que debe compilarse (sin la extensión .vb).

Para crear el programa `DbAuth` a partir del archivo fuente `DbAuth.vb`, entre:

```
bldapp DbAuth
```

Para asegurarse de que tenga los parámetros que necesitará cuando ejecute el archivo ejecutable, puede especificar distintas combinaciones de parámetros en función del número que haya entrado:

1. Ningún parámetro. Entre sólo el nombre del programa:

```
DbAuth
```

2. Un parámetro. Entre el nombre del programa más el alias de la base de datos:
   ```
   DbAuth <alias_bd>
   ```
3. Dos parámetros. Entre el nombre del programa más el ID de usuario y la contraseña:
   ```
   DbAuth <id_usuario> <contraseña>
   ```
4. Tres parámetros. Entre el nombre del programa más el alias de la base de datos, el ID de usuario y la contraseña:
   ```
   DbAuth <alias_bd> <id_usuario> <contraseña>
   ```
5. Cuatro parámetros. Entre el nombre del programa más el nombre del servidor, el número de puerto, el ID de usuario y la contraseña:
   ```
   DbAuth <servidor> <número_puerto> <id_usuario> <contraseña>
   ```
6. Cinco parámetros. Entre el nombre del programa más el alias de la base de datos, el nombre del servidor, el número de puerto, el ID de usuario y la contraseña:
   ```
   DbAuth <alias_bd> <servidor> <número_puerto> <id_usuario> <contraseña>
   ```

Para crear y ejecutar el programa de ejemplo de `LCTrans` debe seguir instrucciones más detalladas que se proporcionan en el archivo fuente, `LCTrans.vb`.

Creación de aplicaciones C# .NET

DB2 proporciona un archivo de proceso por lotes, bldapp.bat, para compilar y vincular las aplicaciones C# .NET de DB2, ubicadas en el directorio sqllib\samples\.NET\cs, junto a los programas de ejemplo que pueden crearse con este archivo. El archivo de proceso por lotes toma un parámetro, %1, para el nombre del archivo fuente que debe compilarse (sin la extensión .cs).

Para asegurarse de que tenga los parámetros que necesitará cuando ejecute el archivo ejecutable, puede especificar distintas combinaciones de parámetros en función del número que haya entrado:

Para crear el programa `DbAuth` a partir del archivo fuente `DbAuth.cs`, entre:

```
bldapp DbAuth
```

Para asegurarse de que tenga los parámetros que necesitará cuando ejecute el archivo ejecutable, puede especificar distintas combinaciones de parámetros en función del número que haya entrado:

1. Ningún parámetro. Entre sólo el nombre del programa:
   ```
   DbAuth
   ```
2. Un parámetro. Entre el nombre del programa más el alias de la base de datos:
   ```
   DbAuth <alias_bd>
   ```
3. Dos parámetros. Entre el nombre del programa más el ID de usuario y la contraseña:
   ```
   DbAuth <id_usuario> <contraseña>
   ```
4. Tres parámetros. Entre el nombre del programa más el alias de la base de datos, el ID de usuario y la contraseña:
   ```
   DbAuth <alias_bd> <id_usuario> <contraseña>
   ```
5. Cuatro parámetros. Entre el nombre del programa más el nombre del servidor, el número de puerto, el ID de usuario y la contraseña:
   ```
   DbAuth <servidor> <número_puerto> <id_usuario> <contraseña>
   ```

6. Cinco parámetros. Entre el nombre del programa más el alias de la base de datos, el nombre del servidor, el número de puerto, el ID de usuario y la contraseña:

```
DbAuth <alias_bd> <servidor> <número_puerto> <id_usuario> <contraseña>
```

Para crear y ejecutar el programa de ejemplo de `LCTrans` debe seguir instrucciones más detalladas que se proporcionan en el archivo fuente, `LCTrans.cs`.

Opciones de compilación y enlace para aplicaciones Visual Basic .NET

La tabla siguiente muestra las opciones de compilación y enlace que DB2 recomienda para crear aplicaciones de Visual Basic .NET en Windows con el compilador Microsoft Visual Basic .NET, tal como muestra el archivo de proceso por lotes `bldapp.bat`.

Opciones de compilación y enlace para bldapp
Opciones de compilación y enlace para aplicaciones VB .NET autónomas: **%BLDCOMP%** Variable del compilador. El valor por omisión es `vbc`, que es el compilador de Microsoft Visual Basic .NET. **/r:"%DB2PATH%"\bin\%VERSION%IBM.Data.DB2.dll** Referencia a la biblioteca de enlaces dinámicos de DB2 para la versión de infraestructura .NET que está utilizando. **%VERSION%** Existen tres versiones de infraestructura de .NET a las que se da soporte para las aplicaciones. DB2 tiene una biblioteca de enlaces dinámicos para cada una de ellas. Para .NET Framework versión 1.1, `%VERSION%` apunta al subdirectorio `netf11\`. Para .NET Framework versión 2.0 y 3.0, `%VERSION%` apunta al subdirectorio `netf20\`.

Opciones de compilación y enlace para el programa de ejemplo emparejado débilmente, LCTrans:

%BLDCOMP%
Variable del compilador. El valor por omisión es `vbc`, que es el compilador de Microsoft Visual Basic .NET.

/out:RootCOM.dll
Salida de la biblioteca de enlaces dinámicos `RootCOM`, utilizada por la aplicación `LCTrans`, del archivo fuente `RootCOM.vb`,

/out:SubCOM.dll
Salida de la biblioteca de enlaces dinámicos `SubCOM`, utilizada por la aplicación `LCTrans`, del archivo fuente `SubCOM.vb`,

/target:library %1.cs
Crear la biblioteca de enlaces dinámicos del archivo fuente de entrada (`RootCOM.vb` o `SubCOM.vb`).

/r:System.EnterpriseServices.dll
Referencia a la biblioteca de enlace de datos de Microsoft Windows System Enterprise Services.

/r:"%DB2PATH%"\bin\%VERSION%IBM.Data.DB2.dll
Referencia a la biblioteca de enlaces dinámicos de DB2 para la versión de infraestructura .NET que está utilizando.

%VERSION%
Existen tres versiones de infraestructura de .NET a las que se da soporte para las aplicaciones. DB2 tiene una biblioteca de enlaces dinámicos para cada una de ellas en subdirectorios independientes. Para .NET Framework versión 1.1, `%VERSION%` apunta al subdirectorio `netf11\`. Para .NET Framework versión 2.0 y 3.0, `%VERSION%` apunta al subdirectorio `netf20\`.

/r:System.Data.dll
Referencia a la biblioteca de enlaces dinámicos de datos de sistema Microsoft Windows.

/r:System.dll
Referencia a la biblioteca de enlaces dinámicos de sistema Microsoft Windows.

/r:System.Xml.dll
Referencia a la biblioteca de enlaces dinámicos XML de sistema Microsoft Windows (para `SubCOM.vb`).

/r:SubCOM.dll
Referencia a la biblioteca de enlaces dinámicos SubCOM (para `RootCOM.vb` y `LCTrans.vb`).

/r:RootCOM.dll
Referencia a la biblioteca de enlaces dinámicos RootCOM (para `LCTrans.vb`).

Consulte la documentación del compilador para conocer otras opciones de compilador.

Opciones de compilación y enlace para aplicaciones C# .NET

La tabla siguiente muestra las opciones de compilación y enlace que DB2 recomienda para crear aplicaciones de C# en Windows con el compilador Microsoft C#, tal como muestra el archivo de proceso por lotes `bldapp.bat`.

Opciones de compilación y enlace para bldapp

Opciones de compilación y enlace para aplicaciones C# autónomas:

%BLDCOMP%
Variable del compilador. El valor por omisión es `csc`, que es el compilador C# de Microsoft.

/r:"%DB2PATH%"\bin\%VERSION%IBM.Data.DB2.dll
Referencia a la biblioteca de enlaces dinámicos de DB2 para la versión de infraestructura .NET que está utilizando.

%VERSION%
Existen tres versiones de infraestructura de .NET a las que se da soporte para las aplicaciones. DB2 tiene una biblioteca de enlaces dinámicos para cada una de ellas en subdirectorios independientes. Para .NET Framework versión 1.1, `%VERSION%` apunta al subdirectorio `netf11\`. Para .NET Framework versión 2.0 y 3.0, `%VERSION%` apunta al subdirectorio `netf20\`.

Opciones de compilación y enlace para el programa de ejemplo emparejado débilmente, LCTrans:

%BLDCOMP%
Variable del compilador. El valor por omisión es csc, que es el compilador C# de Microsoft.

/out:RootCOM.dll
Salida de la biblioteca de enlaces dinámicos RootCOM, utilizada por la aplicación LCTrans, del archivo fuente RootCOM.cs,

/out:SubCOM.dll
Salida de la biblioteca de enlaces dinámicos SubCOM, utilizada por la aplicación LCTrans, del archivo fuente SubCOM.cs,

/target:library %1.cs
Crear la biblioteca de enlaces dinámicos desde el archivo fuente de entrada (RootCOM.cs o SubCOM.cs).

/r:System.EnterpriseServices.dll
Referencia a la biblioteca de enlace de datos de Microsoft Windows System Enterprise Services.

/r:"%DB2PATH%"\bin\%VERSION%IBM.Data.DB2.dll
Referencia a la biblioteca de enlaces dinámicos de DB2 para la versión de infraestructura .NET que está utilizando.

%VERSION%
Existen tres versiones de infraestructura de .NET a las que se da soporte para las aplicaciones. DB2 tiene una biblioteca de enlaces dinámicos para cada una de ellas en subdirectorios independientes. Para .NET Framework versión 1.1, %VERSION% apunta al subdirectorio netf11\. Para .NET Framework versión 2.0 y 3.0, %VERSION% apunta al subdirectorio netf20\.

/r:System.Data.dll
Referencia a la biblioteca de enlaces dinámicos de datos de sistema Microsoft Windows.

/r:System.dll
Referencia a la biblioteca de enlaces dinámicos de sistema Microsoft Windows.

/r:System.Xml.dll
Referencia a la biblioteca de enlaces dinámicos XML de sistema Microsoft Windows (para SubCOM.cs).

/r:SubCOM.dll
Referencia a la biblioteca de enlaces dinámicos SubCOM (para RootCOM.cs y LCTrans.cs).

/r:RootCOM.dll
Referencia a la biblioteca de enlaces dinámicos RootCOM (para LCTrans.cs).

Consulte la documentación del compilador para conocer otras opciones de compilador.

Capítulo 5. IBM OLE DB Provider para DB2

IBM OLE DB Provider para DB2 permite que DB2 actúe como gestor de recursos para OLE DB Provider. Este soporte ofrece a las aplicaciones basadas en OLE DB la posibilidad de extraer o consultar datos de DB2 mediante la interfaz OLE.

Microsoft OLE DB es un conjunto de interfaces OLE/COM que proporciona a las aplicaciones un acceso uniforme a datos almacenados en distintas fuentes de información. La arquitectura OLE DB define a los consumidores de OLE DB y a los proveedores de OLE DB. Un consumidor de OLE DB puede ser cualquier sistema o aplicación que utiliza interfaces OLE DB; un proveedor de OLE DB es un componente que expone las interfaces OLE DB.

IBM OLE DB Provider para DB2, cuyo nombre de proveedor es IBMDADB2, permite a los consumidores de OLE DB acceder a datos en un servidor de bases de datos de DB2. Si DB2 Connect está instalado, estos consumidores de OLE DB también podrán acceder a datos contenidos en sistemas principales DBMS, tales como DB2 para MVS, DB2 para VM/VSE o SQL/400.

IBM OLE DB Provider para DB2 ofrece las siguientes funciones:

- Nivel de soporte 0 de la especificación de proveedor de OLE DB, incluidas algunas interfaces adicionales de nivel 1.
- Una implantación del proveedor de hebras libres, que permite a la aplicación crear componentes en una hebra y utilizar dichos componentes en cualquier otra hebra.
- Un Servicio de búsqueda de errores que devuelve mensajes de error de DB2.

Tenga en cuenta que IBM OLE DB Provider reside en el cliente, y es distinto de las funciones de tabla OLE DB, que también reciben soporte de sistemas de bases de datos de DB2.

Las siguientes secciones de este documento describen la implantación específica de IBM OLE DB Provider para DB2. Para obtener más información sobre la especificación OLE DB 2.0 de Microsoft, consulte el manual "Microsoft OLE DB 2.0 Programmer's Reference and Data Access SDK", publicado por Microsoft Press.

Cumplimiento de las versiones

IBM OLE DB Provider para DB2 se ajusta a la versión 2.7 o posterior de la especificación OLE DB de Microsoft.

Requisitos del sistema

Consulte la carta de presentación correspondiente a IBM OLE DB Provider para servidores DB2 para ver qué sistemas operativos Windows tienen soporte.

Para instalar IBM OLE DB Provider para DB2, debe ejecutar primero uno de los sistemas operativos soportados mencionados anteriormente. También necesita instalar el Cliente DB2. Este cliente incluye Microsoft Data Access Components (MDAC).

Tipos de aplicación soportados por IBM OLE DB Provider para DB2

Con IBM OLE DB Provider para DB2, puede crear los siguientes tipos de aplicaciones:

- Aplicaciones ADO, que incluyen:
  - Aplicaciones Microsoft Visual Studio C++
  - Aplicaciones Microsoft Visual Basic
- Aplicaciones ADO.NET que hacen uso de OLE DB .NET Data Provider
- Aplicaciones C/C++ que acceden a IBMDADB2 directamente mediante las interfaces OLE DB, incluidas aplicaciones ATL cuyos Objetos de consumidor de acceso a datos se han generado mediante ATL COM AppWizard.

Servicios OLE DB

Modelo de hebra soportado por IBM OLE DB Provider

IBM OLE DB Provider para DB2 da soporte al modelo de hebras libres. Esto permite a las aplicaciones crear componentes en una hebra y utilizar estos componentes en cualquier otra hebra.

Manipulación de objetos grandes con IBM OLE DB Provider

Para obtener y establecer datos como objetos de almacenamiento (`DBTYPE_IUNKNOWN`) con el proveedor de IBMDADB2, utilice la interfaz de ISequentialStream de la forma siguiente:

- Para vincular un objeto de almacenamiento a un parámetro, DBOBJECT en la estructura DBBINDING sólo puede contener el valor `STGM_READ` para el campo dwFlag. IBMDADB2 ejecutará el método Read de la interfaz ISequentialStream del objeto vinculado.
- Para obtener datos de un objeto de almacenamiento, la aplicación debe ejecutar el método Read en la interfaz ISequentialStream del objeto de almacenamiento.
- Cuando se obtienen datos, el valor de la parte de longitud es la longitud de los datos reales, no la longitud del puntero IUnknown.

Conjuntos de filas de esquema soportados por IBM OLE DB Provider

La tabla siguiente muestra los conjuntos de filas de esquema soportados por `IDBSchemaRowset`. Las columnas no soportadas se establecerán en `nulo` en los conjuntos de filas.

Tabla 29. Conjuntos de filas de esquema soportados por IBM OLE DB Provider para DB2

GUID soportados	Restricciones soportadas	Columnas soportadas	Notas
DBSCHEMA _COLUMN_PRIVILEGES	COLUMN_NAME TABLE_NAME TABLE_SCHEMA	COLUMN_NAME GRANTEE GRANTOR IS_GRANTABLE PRIVILEGE_TYPE TABLE_NAME TABLE_SCHEMA	

Tabla 29. Conjuntos de filas de esquema soportados por IBM OLE DB Provider para DB2 (continuación)

GUID soportados	Restricciones soportadas	Columnas soportadas	Notas
DBSCHEMA_COLUMNS	COLUMN_NAME TABLE_NAME TABLE_SCHEMA	CHARACTER_MAXIMUM_LENGTH CHARACTER_OCTET_LENGTH COLUMN_DEFAULT COLUMN_FLAGS COLUMN_HASDEFAULT COLUMN_NAME DATA_TYPE DESCRIPTION IS_NULLABLE NUMERIC_PRECISION NUMERIC_SCALE ORDINAL_POSITION TABLE_NAME TABLE_SCHEMA	
DBSCHEMA_FOREIGN_KEYS	FK_TABLE_NAME FK_TABLE_SCHEMA PK_TABLE_NAME PK_TABLE_SCHEMA	DEFERRABILITY DELETE_RULE FK_COLUMN_NAME FK_NAME FK_TABLE_NAME FK_TABLE_SCHEMA ORDINAL PK_COLUMN_NAME PK_NAME PK_TABLE_NAME PK_TABLE_SCHEMA UPDATE_RULE	Se debe especificar al menos una de las siguientes restricciones: PK_TABLE_NAME o FK_TABLE_NAME No se permite el carácter comodín "%".
DBSCHEMA_INDEXES	TABLE_NAME TABLE_SCHEMA	CARDINALITY CLUSTERED COLLATION COLUMN_NAME INDEX_NAME INDEX_SCHEMA ORDINAL_POSITION PAGES TABLE_NAME TABLE_SCHEMA TYPE UNIQUE	No se permite ningún orden de clasificación. El orden de clasificación, si se especifica, se pasará por alto.
DBSCHEMA_PRIMARY_KEYS	TABLE_NAME TABLE_SCHEMA	COLUMN_NAME ORDINAL PK_NAME TABLE_NAME TABLE_SCHEMA	Se debe especificar al menos la siguiente restricción: TABLE_NAME No se permite el carácter comodín "%".

Tabla 29. Conjuntos de filas de esquema soportados por IBM OLE DB Provider para DB2 (continuación)

GUID soportados	Restricciones soportadas	Columnas soportadas	Notas
DBSCHEMA _PROCEDURE_PARAMETERS	PARAMETER_NAME PROCEDURE_NAME PROCEDURE_SCHEMA	CHARACTER_MAXIMUM_LENGTH CHARACTER_OCTET_LENGTH DATA_TYPE DESCRIPTION IS_NULLABLE NUMERIC_PRECISION NUMERIC_SCALE ORDINAL_POSITION PARAMETER_DEFAULT PARAMETER_HASDEFAULT PARAMETER_NAME PARAMETER_TYPE PROCEDURE_NAME PROCEDURE_SCHEMA TYPE_NAME	
DBSCHEMA_PROCEDURES	PROCEDURE_NAME PROCEDURE_SCHEMA	DESCRIPTION PROCEDURE_NAME PROCEDURE_SCHEMA PROCEDURE_TYPE	
DBSCHEMA_PROVIDER_TYPES	DATA_TYPE BEST_MATCH	AUTO_UNIQUE_VALUE BEST_MATCH CASE_SENSITIVE CREATE_PARAMS COLUMN_SIZE DATA_TYPE FIXED_PREC_SCALE IS_FIXEDLENGTH IS_LONG IS_NULLABLE LITERAL_PREFIX LITERAL_SUFFIX LOCAL_TYPE_NAME MINIMUM_SCALE MAXIMUM_SCALE SEARCHABLE TYPE_NAME UNSIGNED_ATTRIBUTE	
DBSCHEMA_STATISTICS	TABLE_NAME TABLE_SCHEMA	CARDINALITY TABLE_NAME TABLE_SCHEMA	No se permite ningún orden de clasificación. El orden de clasificación, si se especifica, se pasará por alto.
DBSCHEMA _TABLE_PRIVILEGES	TABLE_NAME TABLE_SCHEMA	GRANTEE GRANTOR IS_GRANTABLE PRIVILEGE_TYPE TABLE_NAME TABLE_SCHEMA	
DBSCHEMA_TABLES	TABLE_NAME TABLE_SCHEMA TABLE_TYPE	DESCRIPTION TABLE_NAME TABLE_SCHEMA TABLE_TYPE	

Habilitación automática de servicios OLE DB por parte de IBM OLE DB Provider

Por omisión, IBM OLE DB Provider para DB2 habilita automáticamente todos los servicios OLE DB añadiendo una entrada de registro `OLEDB_SERVICES` al ID de clase (CLSID) del proveedor con `0xFFFFFFFF` como valor de DWORD. El significado de este valor es el siguiente:

Tabla 30. Servicios OLE DB

Servicios habilitados	Valor de DWORD
Todos los servicios (valor por omisión)	`0xFFFFFFFF`
Todos excepto agrupación y AutoEnlistment	`0xFFFFFFFC`
Todos excepto cursor del cliente	`0xFFFFFFFB`
Todos excepto agrupación, alistamiento y cursor	`0xFFFFFFF8`
Ningún servicio	`0x000000000`

Servicios de datos

Modalidades de cursor soportadas para IBM OLE DB Provider

IBM OLE DB Provider para DB2 da soporte nativo a cursores de sólo lectura, de sólo avance, así como actualizables y desplazables.

Correlaciones de tipos de datos entre DB2 y OLE DB

IBM OLE DB Provider para DB2 da soporte a correlaciones de tipos de datos entre tipos de datos de DB2 y tipos de datos OLE DB.

La tabla siguiente proporciona una lista completa de correlaciones soportadas y nombres disponibles para indicar los tipos de datos de columnas y parámetros.

Tabla 31. Correlaciones de tipos de datos DB2 y tipos de datos OLE DB

Tipos de datos DB2	Indicadores de tipos de datos OLE DB	Nombres de tipos estándar de OLE DB	Nombres específicos de DB2
SMALLINT	DBTYPE_I2	"DBTYPE_I2"	"SMALLINT"
INTEGER	DBTYPE_I4	"DBTYPE_I4"	"INTEGER" o "INT"
BIGINT	DBTYPE_I8	"DBTYPE_I8"	"BIGINT"
REAL	DBTYPE_R4	"DBTYPE_R4"	"REAL"
FLOAT	DBTYPE_R8	"DBTYPE_R8"	"FLOAT"
DOUBLE	DBTYPE_R8	"DBTYPE_R8"	"DOUBLE" o "DOUBLE PRECISION"
DECIMAL	DBTYPE_NUMERIC	"DBTYPE_NUMERIC"	"DEC" o "DECIMAL"
NUMERIC	DBTYPE_NUMERIC	"DBTYPE_NUMERIC"	"NUM" o "NUMERIC"
DATE	DBTYPE_DBDATE	"DBTYPE_DBDATE"	"DATE"
TIME	DBTYPE_DBTIME	"DBTYPE_DBTIME"	"TIME"
TIMESTAMP	DBTYPE_DBTIMESTAMP	"DBTYPE_DBTIMESTAMP"	"TIMESTAMP"
CHAR	DBTYPE_STR	"DBTYPE_CHAR"	"CHAR" o "CHARACTER"

Tabla 31. Correlaciones de tipos de datos DB2 y tipos de datos OLE DB (continuación)

Tipos de datos DB2	Indicadores de tipos de datos OLE DB	Nombres de tipos estándar de OLE DB	Nombres específicos de DB2
VARCHAR	DBTYPE_STR	"DBTYPE_VARCHAR"	"VARCHAR"
LONG VARCHAR	DBTYPE_STR	"DBTYPE_LONGVARCHAR"	"LONG VARCHAR"
CLOB	DBTYPE_STR y DBCOLUMNFLAGS_ISLONG o DBPARAMFLAGS_ISLONG	"DBTYPE_CHAR" "DBTYPE_VARCHAR" "DBTYPE_LONGVARCHAR" y DBCOLUMNFLAGS_ISLONG o DBPARAMFLAGS_ISLONG	"CLOB"
GRAPHIC	DBTYPE_WSTR	"DBTYPE_WCHAR"	"GRAPHIC"
VARGRAPHIC	DBTYPE_WSTR	"DBTYPE_WVARCHAR"	"VARGRAPHIC"
LONG VARGRAPHIC	DBTYPE_WSTR	"DBTYPE_WLONGVARCHAR"	"LONG VARGRAPHIC"
DBCLOB	DBTYPE_WSTR y DBCOLUMNFLAGS_ISLONG o DBPARAMFLAGS_ISLONG	"DBTYPE_WCHAR" "DBTYPE_WVARCHAR" "DBTYPE_WLONGVARCHAR" y DBCOLUMNFLAGS_ISLONG o DBPARAMFLAGS_ISLONG	"DBCLOB"
CHAR(n) FOR BIT DATA	DBTYPE_BYTES	"DBTYPE_BINARY"	
VARCHAR(n) FOR BIT DATA	DBTYPE_BYTES	"DBTYPE_VARBINARY"	
LONG VARCHAR FOR BIT DATA	DBTYPE_BYTES	"DBTYPE_LONGVARBINARY"	
BLOB	DBTYPE_BYTES y DBCOLUMNFLAGS_ISLONG o DBPARAMFLAGS_ISLONG	"DBTYPE_BINARY" "DBTYPE_VARBINARY" "DBTYPE_LONGVARBINARY" y DBCOLUMNFLAGS_ISLONG o DBPARAMFLAGS_ISLONG	"BLOB"

Conversión de datos para establecer datos de tipos OLE DB en tipos DB2

IBM OLE DB Provider para DB2 da soporte a las conversiones para establecer datos de tipos OLE DB en tipos DB2.

La tabla siguiente muestra conversiones de datos de tipos OLE DB en tipos DB2. Tenga en cuenta que es posible que se trunquen datos en algunos casos, en función de los tipos y del valor de los datos.

Tabla 32. Conversiones de datos de tipos OLE DB a tipos DB2

Indicador de tipo OLE DB	Tipos de datos DB2																						
																		For Bit Data					
	SMALLINT	INTEGER	BIGINT	REAL	FLOAT DOUBLE	DECIMAL NUMERIC	DATE	TIME	TIMESTAMP	CHAR	VARCHAR	LONG VARCHAR	CLOB	GRAPHIC	VARGRAPHIC	LONG VARGRAPHIC	DBCLOB	CHAR	VARCHAR	LONG VARCHAR	BLOB	DATALINK
DBTYPE_EMPTY																							
DBTYPE_NULL																							
DBTYPE_RESERVED																							
DBTYPE_I1	X	X	X	X	X	X				X	X												
DBTYPE_I2	X	X	X	X	X	X				X	X												
DBTYPE_I4	X	X	X	X	X	X				X	X												
DBTYPE_I8	X	X	X	X	X	X				X	X												
DBTYPE_UI1	X	X	X	X	X	X				X	X												
DBTYPE_UI2	X	X	X	X	X	X				X	X												
DBTYPE_UI4	X	X	X	X	X	X				X	X												
DBTYPE_UI8	X	X	X	X	X	X				X	X												
DBTYPE_R4	X	X	X	X	X	X				X	X												
DBTYPE_R8	X	X	X	X	X	X				X	X												
DBTYPE_CY																							
DBTYPE_DECIMAL	X	X	X	X	X	X				X	X												
DBTYPE_NUMERIC	X	X	X	X	X	X				X	X												

Tabla 32. Conversiones de datos de tipos OLE DB a tipos DB2 (continuación)

Indicador de tipo OLE DB	Tipos de datos DB2																						
																			For Bit Data				
	SMALLINT	INTEGER	BIGINT	REAL	FLOAT DOUBLE	DECIMAL NUMERIC	DATE	TIME	TIMESTAMP	CHAR	VARCHAR	LONG VARCHAR	CLOB	GRAPHIC	VARGRAPHIC	LONG VARGRAPHIC	DBCLOB	CHAR	VARCHAR	LONG VARCHAR	BLOB	DATALINK
DBTYPE_DATE																							
DBTYPE_BOOL	X	X	X	X	X	X				X	X												
DBTYPE_BYTES			X			X				X	X	X				X		X	X	X			
DBTYPE_BSTR – por determinar																							
DBTYPE_STR	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X		X	X	X		X	
DBTYPE_WSTR														X	X	X							
DBTYPE_VARIANT – por determinar																							
DBTYPE_IDISPATCH																							
DBTYPE_IUNKNOWN										X	X	X	X	X	X	X	X	X	X	X			
DBTYPE_GUID																							
DBTYPE_ERROR																							
DBTYPE_BYREF																							
DBTYPE_ARRAY																							
DBTYPE_VECTOR																							
DBTYPE_UDT																							

Tabla 32. Conversiones de datos de tipos OLE DB a tipos DB2 (continuación)

Indicador de tipo OLE DB	Tipos de datos DB2																						
	SMALLINT	INTEGER	BIGINT	REAL	FLOAT DOUBLE	DECIMAL NUMERIC	DATE	TIME	TIMESTAMP	CHAR	VARCHAR	LONG VARCHAR	CLOB	GRAPHIC	VARGRAPHIC	LONG VARGRAPHIC	DBCLOB	For Bit Data			BLOB	DATALINK
																		CHAR	VARCHAR	LONG VARCHAR		
DBTYPE_DBDATE							X		X	X	X											
DBTYPE_DBTIME								X	X	X	X											
DBTYPE_DBTIMESTAMP							X	X	X	X	X											
DBTYPE_FILETIME																						
DBTYPE_PROP_VARIANT																						
DBTYPE_HCHAPTER																						
DBTYPE_VARNUMERIC																						

Conversión de datos para establecer datos de tipos DB2 en tipos OLE DB

Para obtener datos, IBM OLE DB Provider permite realizar conversiones de datos de tipos DB2 a tipos OLE DB.

En la tabla siguiente se muestran las conversiones de datos soportadas de tipos DB2 en tipos OLE DB. Tenga en cuenta que es posible que se trunquen datos en algunos casos, en función de los tipos y del valor de los datos.

Tabla 33. Conversiones de datos de tipos DB2 a tipos OLE DB

	Tipos de datos DB2																					
																		For Bit Data				
Indicador de tipo OLE DB	**SMALLINT**	**INTEGER**	**BIGINT**	**REAL**	**FLOAT DOUBLE**	**DECIMAL NUMERIC**	**DATE**	**TIME**	**TIMESTAMP**	**CHAR**	**VARCHAR**	**LONG VARCHAR**	**CLOB**	**GRAPHIC**	**VARGRAPHIC**	**LONG VARGRAPHIC**	**DBCLOB**	**CHAR**	**VARCHAR**	**LONG VARCHAR**	**BLOB**	**DATALINK**
DBTYPE_EMPTY																						
DBTYPE_NULL																						
DBTYPE_RESERVED																						
DBTYPE_I1	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_I2	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_I4	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_I8	X	X	X	X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_UI1	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_UI2	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_UI4	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_UI8	X	X	X	X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_R4	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_R8	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_CY	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_DECIMAL	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_NUMERIC	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X

Tabla 33. Conversiones de datos de tipos DB2 a tipos OLE DB (continuación)

Indicador de tipo OLE DB	Tipos de datos DB2																						
																			For Bit Data				
	SMALLINT	INTEGER	BIGINT	REAL	FLOAT DOUBLE	DECIMAL NUMERIC	DATE	TIME	TIMESTAMP	CHAR	VARCHAR	LONG VARCHAR	CLOB	GRAPHIC	VARGRAPHIC	LONG VARGRAPHIC	DBCLOB	CHAR	VARCHAR	LONG VARCHAR	BLOB	DATALINK
DBTYPE_DATE	X	X		X	X		X	X	X	X	X	X		X	X	X						X
DBTYPE_BOOL	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_BYTES	X	X		X	X	X	X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_BSTR	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_STR	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_WSTR	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_VARIANT	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_IDISPATCH																						
DBTYPE_IUNKNOWN	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
DBTYPE_GUID										X	X	X		X	X	X		X	X	X		X
DBTYPE_ERROR																						
DBTYPE_BYREF																						
DBTYPE_ARRAY																						
DBTYPE_VECTOR																						
DBTYPE_UDT																						
DBTYPE_DBDATE							X	X	X	X	X	X		X	X	X		X	X	X		X

Tabla 33. Conversiones de datos de tipos DB2 a tipos OLE DB (continuación)

Indicador de tipo OLE DB	Tipos de datos DB2																					
																		For Bit Data				
	SMALLINT	INTEGER	BIGINT	REAL	FLOAT DOUBLE	DECIMAL NUMERIC	DATE	TIME	TIMESTAMP	CHAR	VARCHAR	LONG VARCHAR	CLOB	GRAPHIC	VARGRAPHIC	LONG VARGRAPHIC	DBCLOB	CHAR	VARCHAR	LONG VARCHAR	BLOB	DATALINK
DBTYPE_DBTIME							X	X	X	X	X	X		X	X	X						X
DBTYPE_DBTIMESTAMP							X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_FILETIME			X				X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_PROP_VARIANT	X	X	X	X	X					X	X	X		X	X	X		X	X	X		X
DBTYPE_HCHAPTER																						
DBTYPE_VARNUMERIC																						

Nota: Cuando la aplicación realiza una operación ISequentialStream::Read para obtener los datos del objeto de almacenamiento, el formato de los datos devueltos depende del tipo de datos de la columna:

- Para tipos de datos binarios y que no sean de tipo carácter, los datos de la columna se exponen como una secuencia de bytes que representan dichos valores en el sistema operativo.
- Para tipos de datos de carácter, primero los datos se convierten a DBTYPE_STR.
- Para DBCLOB, primero los datos se convierten a DBTYPE_WCHAR.

Restricciones de IBM OLE DB Provider

A continuación se describen las restricciones de IBM OLE DB Provider:

- IBMDADB2 da soporte al ámbito de transacciones de confirmación automática y controladas por el usuario con la interfaz `ITransactionLocal`. El ámbito de transacciones de confirmación automática es el ámbito por omisión. Las transacciones anidadas no reciben soporte.
- `RestartPosition` no recibe soporte si el texto del mandato contiene parámetros.
- IBMDADB2 no pone entre comillas nombres de tablas que se pasan a través de parámetros de DBID, que son los parámetros que utiliza la interfaz `IOpenRowset`. En su lugar, el consumidor de OLE DB debe añadir comillas a los nombres de tablas cuando sea necesario.

Soporte de IBM OLE DB para componentes e interfaces de OLE DB

En la tabla siguiente se indican los componentes y las interfaces de OLE DB a las que IBM OLE DB Provider para DB2 y Microsoft OLE DB Provider para ODBC dan soporte.

Tabla 34. BLOB

Interfaz	DB2	ODBC Provider
ISequentialStream	Sí	Sí

Tabla 35. Mandatos

Interfaz	DB2	ODBC Provider
IAccessor	Sí	Sí
ICommand	Sí	Sí
ICommandPersist	No	No
ICommandPrepare	Sí	Sí
ICommandProperties	Sí	Sí
ICommandText	Sí	Sí
ICommandWithParameters	Sí	Sí
IColumnsInfo	Sí	Sí
IColumnsRowset	Sí	Sí
IConvertType	Sí	Sí
ISupportErrorInfo	Sí	Sí

Tabla 36. Fuentes de datos

Interfaz	DB2	ODBC Provider
IConnectionPoint	No	Sí
IDBAsynchNotify (consumidor)	No	No
IDBAsynchStatus	No	No
IDBConnectionPointContainer	No	Sí
IDBCreateSession	Sí	Sí
IDBDataSourceAdmin	No	No
IDBInfo	Sí	Sí
IDBInitialize	Sí	Sí
IDBProperties	Sí	Sí
IPersist	Sí	No
IPersistFile	Sí	Sí
ISupportErrorInfo	Sí	Sí

Tabla 37. Enumeradores

Interfaz	DB2	ODBC Provider
IDBInitialize	Sí	Sí
IDBProperties	Sí	Sí
IParseDisplayName	Sí	No

Tabla 37. Enumeradores (continuación)

Interfaz	DB2	ODBC Provider
ISourcesRowset	Sí	Sí
ISupportErrorInfo	Sí	Sí

Tabla 38. Servicio de búsqueda de errores

Interfaz	DB2	ODBC Provider
IErrorLookUp	Sí	Sí

Tabla 39. Objetos de errores

Interfaz	DB2	ODBC Provider
IErrorInfo	Sí	No
	Sí	No
ISQLErrorInfo (personalizado)	Sí	No

Tabla 40. Varios resultados

Interfaz	DB2	ODBC Provider
IMultipleResults	Sí	Sí
ISupportErrorInfo	Sí	Sí

Tabla 41. Conjuntos de filas

Interfaz	DB2	ODBC Provider
IAccessor	Sí	Sí
IColumnsRowset	Sí	Sí
IColumnsInfo	Sí	Sí
IConvertType	Sí	Sí
IChapteredRowset	No	No
IConnectionPointContainer	Sí	Sí
IDBAsynchStatus	No	No
IParentRowset	No	No
IRowset	Sí	Sí
IRowsetChange	Sí	Sí
IRowsetChapterMember	No	No
IRowsetFind	No	No
IRowsetIdentity	Sí	Sí
IRowsetIndex	No	No
IRowsetInfo	Sí	Sí
IRowsetLocate	Sí	Sí
IRowsetNotify (consumidor)	Sí	No
IRowsetRefresh	Componente Servicio de cursor	Sí
IRowsetResynch	Componente Servicio de cursor	Sí
IRowsetScroll	Sí[1]	Sí

Tabla 41. Conjuntos de filas (continuación)

Interfaz	DB2	ODBC Provider
IRowsetUpdate	Componente Servicio de cursor	Sí
IRowsetView	No	No
ISupportErrorInfo	Sí	Sí
Nota: 1. Los valores a devolver son aproximaciones. Las filas suprimidas no se pasarán por alto.		

Tabla 42. Sesiones

Interfaz	DB2	ODBC Provider
IAlterIndex	No	No
IAlterTable	No	No
IDBCreateCommand	Sí	Sí
IDBSchemaRowset	Sí	Sí
IGetDataSource	Sí	Sí
IIndexDefinition	No	No
IOpenRowset	Sí	Sí
ISessionProperties	Sí	Sí
ISupportErrorInfo	Sí	Sí
ITableDefinition	No	No
ITableDefinitionWithConstraints	No	No
ITransaction	Sí	Sí
ITransactionJoin	Sí	Sí
ITransactionLocal	Sí	Sí
ITransactionObject	No	No
ITransactionOptions	No	Sí

Tabla 43. Objetos Vista

Interfaz	DB2	ODBC Provider
IViewChapter	No	No
IViewFilter	No	No
IViewRowset	No	No
IViewSort	No	No

Soporte de IBM OLE DB Provider para propiedades de OLE DB

La tabla siguiente muestra las propiedades de OLE DB que reciben soporte de IBM OLE DB Provider para DB2:

Tabla 44. Propiedades soportadas por IBM OLE DB Provider para DB2: Origen de datos (DBPROPSET_DATASOURCE)

Propiedades	Valor por omisión	R/W
DBPROP_MULTIPLECONNECTIONS	VARIANT_FALSE	R

Tabla 44. Propiedades soportadas por IBM OLE DB Provider para DB2: Origen de datos (DBPROPSET_DATASOURCE) (continuación)

Propiedades	Valor por omisión	R/W
DBPROP_RESETDATASOURCE	DBPROPVAL_RD_RESETALL	R/W

Tabla 45. Propiedades soportadas por IBM OLE DB Provider para DB2: Origen de datos de DB2 (DBPROPSET_DB2DATASOURCE)

Propiedades	Valor por omisión	R/W
DB2PROP_REPORTISLONGFORLONGTYPES	VARIANT_FALSE	R/W
DB2PROP_RETURNCHARASWCHAR	VARIANT_TRUE	R/W
DB2PROP_SORTBYORDINAL	VARIANT_FALSE	R/W

Tabla 46. Propiedades soportadas por IBM OLE DB Provider para DB2: Información de origen de datos (DBPROPSET_DATASOURCEINFO)

Propiedades	Valor por omisión	R/W
DBPROP_ACTIVESESSIONS	0	R
DBPROP_ASYNCTXNABORT	VARIANT_FALSE	R
DBPROP_ASYNCTXNCOMMIT	VARIANT_FALSE	R
DBPROP_BYREFACCESSORS	VARIANT_FALSE	R
DBPROP_COLUMNDEFINITION	DBPROPVAL_CD_NOTNULL	R
DBPROP_CONCATNULLBEHAVIOR	DBPROPVAL_CB_NULL	R
DBPROP_CONNECTIONSTATUS	DBPROPVAL_CS_INITIALIZED	R
DBPROP_DATASOURCENAME	N/D	R
DBPROP_DATASOURCEREADONLY	VARIANT_FALSE	R
DBPROP_DBMSNAME	N/D	R
DBPROP_DBMSVER	N/D	R
DBPROP_DSOTHREADMODEL	DBPROPVAL_RT_FREETHREAD	R
DBPROP_GROUPBY	DBPROPVAL_GB_CONTAINS_SELECT	R
DBPROP_IDENTIFIERCASE	DBPROPVAL_IC_UPPER	R
DBPROP_MAXINDEXSIZE	0	R
DBPROP_MAXROWSIZE	0	R
DBPROP_MAXROWSIZEINCLUDESBLOB	VARIANT_TRUE	R
DBPROP_MAXTABLEINSELECT	0	R
DBPROP_MULTIPLEPARAMSETS	VARIANT_FALSE	R
DBPROP_MULTIPLERESULTS	DBPROPVAL_MR_SUPPORTED	R
DBPROP_MULTIPLESTORAGEOBJECTS	VARIANT_TRUE	R
DBPROP_MULTITABLEUPDATE	VARIANT_FALSE	R
DBPROP_NULLCOLLATION	DBPROPVAL_NC_LOW	R
DBPROP_OLEOBJECTS	DBPROPVAL_OO_BLOB	R
DBPROP_ORDERBYCOLUMNSINSELECT	VARIANT_FALSE	R
DBPROP _OUTPUTPARAMETERAVAILABILITY	DBPROPVAL_OA_ATEXECUTE	R

Tabla 46. Propiedades soportadas por IBM OLE DB Provider para DB2: Información de origen de datos (DBPROPSET_DATASOURCEINFO) (continuación)

Propiedades	Valor por omisión	R/W
DBPROP_PERSISTENTIDTYPE	DBPROPVAL_PT_NAME	R
DBPROP_PREPAREABORTBEHAVIOR	DBPROPVAL_CB_DELETE	R
DBPROP_PROCEDURETERM	"PROCEDIMIENTO ALMACENADO"	R
DBPROP_PROVIDERFRIENDLYNAME	"IBM OLE DB Provider para DB2"	R
DBPROP_PROVIDERNAME	"IBMDADB2.DLL"	R
DBPROP_PROVIDEROLEDBVER	"02.7"	R
DBPROP_PROVIDERVER	N/D	R
DBPROP_QUOTEIDENTIFIERCASE	DBPROPVAL_IC_SENSITIVE	R
DBPROP _ROWSETCONVERSIONSONCOMMAND	VARIANT_TRUE	R
DBPROP_SCHEMATERM	"ESQUEMA"	R
DBPROP_SCHEMAUSAGE	DBPROPVAL_SU_DML_STATEMENTS | DBPROPVAL_SU_TABLE_DEFINITION | DBPROPVAL_SU_INDEX_DEFINITION | DBPROPVAL_SU_PRIVILEGE_DEFINITION	R
DBPROP_SQLSUPPORT	DBPROPVAL_SQL_ODBC_EXTENDED | DBPROPVAL_SQL_ESCAPECLAUSES | DBPROPVAL_SQL_ANSI92_ENTRY	R
DBPROP_SERVERNAME	N/D	R
DBPROP_STRUCTUREDSTORAGE	DBPROPVAL_SS_ISEQUENTIALSTREAM	R
DBPROP_SUBQUERIES	DBPROPVAL_SQ_CORRELATEDSUBQUERIES | DBPROPVAL_SQ_COMPARISON | DBPROPVAL_SQ_EXISTS | DBPROPVAL_SQ_IN | DBPROPVAL_SQ_QUANTIFIED |	R
DBPROP_SUPPORTEDTXNDDL	DBPROPVAL_TC_ALL	R
DBPROP_SUPPORTEDTXNISOLEVELS	DBPROPVAL_TI_CURSORSTABILITY | DBPROPVAL_TI_READCOMMITTED | DBPROPVAL_TI_READUNCOMMITTED | DBPROPVAL_TI_SERIALIZABLE |	R
DBPROP_SUPPORTEDTXNISORETAIN	DBPROPVAL_TR_COMMIT_DC | DBPROPVAL_TR_ABORT_NO |	R
DBPROP_TABLETERM	"TABLA"	R
DBPROP_USERNAME	N/D	R

Tabla 47. Propiedades soportadas por IBM OLE DB Provider para DB2: Inicialización (DBPROPSET_DBINIT)

Propiedades	Valor por omisión	R/W
DBPROP_AUTH_PASSWORD	N/D	R/W
DBPROP_INIT_TIMEOUT (1)	0	R/W

Tabla 47. Propiedades soportadas por IBM OLE DB Provider para DB2: Inicialización (DBPROPSET_DBINIT) (continuación)

Propiedades	Valor por omisión	R/W
DBPROP_AUTH_PERSIST _SENSITIVE_AUTHINFO	VARIANT_FALSE	R/W
DBPROP_AUTH_USERID	N/D	R/W
DBPROP_INIT_DATASOURCE	N/D	R/W
DBPROP_INIT_HWND	N/D	R/W
DBPROP_INIT_MODE	DB_MODE_READWRITE	R/W
DBPROP_INIT_OLEDBSERVICES	0xFFFFFFFF	R/W
DBPROP_INIT_PROMPT	DBPROMPT_NOPROMPT	R/W
DBPROP_INIT_PROVIDERSTRING	N/D	R/W

Tabla 48. Propiedades soportadas por IBM OLE DB Provider para DB2: Conjunto de filas (DBPROPSET_ROWSET)

Propiedades	Valor por omisión	R/W
DBPROP_ABORTPRESERVE	VARIANT_FALSE	R
DBPROP_ACCESSORDER	DBPROPVAL_AO_RANDOM	R
DBPROP_BLOCKINGSTORAGEOBJECTS	VARIANT_FALSE	R
DBPROP_BOOKMARKS	VARIANT_FALSE	R/W
DBPROP_BOOKMARKSKIPPED	VARIANT_FALSE	R
DBPROP_BOOKMARKTYPE	DBPROPVAL_BMK_NUMERIC	R
DBPROP_CACHEDEFERRED	VARIANT_FALSE	R/W
DBPROP_CANFETCHBACKWARDS	VARIANT_FALSE	R/W
DBPROP_CANHOLDROWS	VARIANT_FALSE	R
DBPROP_CANSCROLLBACKWARDS	VARIANT_FALSE	R/W
DBPROP_CHANGEINSERTEDROWS	VARIANT_FALSE	R
DBPROP_COMMITPRESERVE	VARIANT_TRUE	R/W
DBPROP_COMMANDTIMEOUT	0	R/W
DBPROP_DEFERRED	VARIANT_FALSE	R
DBPROP_IAccessor	VARIANT_TRUE	R
DBPROP_IColumnsInfo	VARIANT_TRUE	R
DBPROP_IColumnsRowset	VARIANT_TRUE	R/W
DBPROP_IConvertType	VARIANT_TRUE	R
DBPROP_IMultipleResults	VARIANT_FALSE	R/W
DBPROP_IRowset	VARIANT_TRUE	R
DBPROP_IRowChange	VARIANT_FALSE	R/W
DBPROP_IRowsetFind	VARIANT_FALSE	R
DBPROP_IRowsetIdentity	VARIANT_TRUE	R
DBPROP_IRowsetInfo	VARIANT_TRUE	R
DBPROP_IRowsetLocate	VARIANT_FALSE	R/W
DBPROP_IRowsetScroll	VARIANT_FALSE	R/W
DBPROP_IRowsetUpdate	VARIANT_FALSE	R

Tabla 48. Propiedades soportadas por IBM OLE DB Provider para DB2: Conjunto de filas (DBPROPSET_ROWSET) (continuación)

Propiedades	Valor por omisión	R/W
DBPROP_ISequentialStream	VARIANT_TRUE	R
DBPROP_ISupportErrorInfo	VARIANT_TRUE	R
DBPROP_LITERALBOOKMARKS	VARIANT_FALSE	R
DBPROP_LITERALIDENTITY	VARIANT_TRUE	R
DBPROP_LOCKMODE	DBPROPVAL_LM_SINGLEROW	R/W
DBPROP_MAXOPENROWS	32767	R
DBPROP_MAXROWS	0	R/W
DBPROP_NOTIFICATIONGRANULARITY	DBPROPVAL_NT_SINGLEROW	R/W
DBPROP_NOTIFICATION PHASES	DBPROPVAL_NP_OKTODO DBPROPBAL_NP_ABOUTTODO DBPROPVAL_NP_SYNCHAFTER DBPROPVAL_NP_FAILEDTODO DBPROPVAL_NP_DIDEVENT	R
DBPROP_NOTIFYROWSETRELEASE	DBPROPVAL_NP_OKTODO DBPROPVAL_NP_ABOUTTODO	R
DBPROP _NOTIFYROWSETFETCHPOSITIONCHANGE	DBPROPVAL_NP_OKTODO DBPROPVAL_NP_ABOUTTODO	R
DBPROP_NOTIFYCOLUMNSET	DBPROPVAL_NP_OKTODO DBPROPVAL_NP_ABOUTTODO	R
DBPROP_NOTIFYROWDELETE	DBPROPVAL_NP_OKTODO DBPROPVAL_NP_ABOUTTODO	R
DBPROP_NOTIFYROWINSERT	DBPROPVAL_NP_OKTODO DBPROPVAL_NP_ABOUTTODO	R
DBPROP_ORDEREDBOOKMARKS	VARIANT_FALSE	R
DBPROP_OTHERINSERT	VARIANT_FALSE	R
DBPROP_OTHERUPDATEDELETE	VARIANT_FALSE	R/W
DBPROP_OWNINSERT	VARIANT_FALSE	R
DBPROP_OWNUPDATEDELETE	VARIANT_FALSE	R
DBPROP_QUICKRESTART	VARIANT_FALSE	R/W
DBPROP_REMOVEDELETED	VARIANT_FALSE	R/W
DBPROP_ROWTHREADMODEL	DBPROPVAL_RT_FREETHREAD	R
DBPROP_SERVERCURSOR	VARIANT_TRUE	R
DBPROP_SERVERDATAONINSERT	VARIANT_FALSE	R
DBPROP_UNIQUEROWS	VARIANT_FALSE	R/W
DBPROP_UPDATABILITY	0	R/W

Tabla 49. Propiedades soportadas por IBM OLE DB Provider para DB2: Conjunto de filas DB2 (DBPROPSET_DB2ROWSET)

Propiedades	Valor por omisión	R/W
DBPROP_ISLONGMINLENGTH	32000	R/W

Tabla 50. Propiedades soportadas por IBM OLE DB Provider para DB2: Sesión (DBPROPSET_SESSION)

Propiedades	Valor por omisión	R/W
DBPROP_SESS_AUTOCOMMITISOLEVELS	DBPROPVAL_TI_CURSORSTABILITY	R/W

Nota:

1. El tiempo de espera excedido sólo puede aplicarse al utilizar el protocolo TCP/IP para conectarse al servidor. El tiempo de espera excedido sólo se aplica durante la conexión de sock de TCP/IP. Si la conexión de sock se completa antes de que caduque el tiempo de espera especificado, el tiempo de espera no podrá aplicarse al resto del proceso de inicialización. Si se utiliza la característica de redireccionamiento-cliente se doblará el tiempo de espera excedido. En general, cuando se habilita el redireccionamiento, dicho redireccionamiento de cliente determinará el comportamiento del tiempo de espera excedido de la conexión.

Conexiones a fuentes de datos mediante IBM OLE DB Provider

La tabla siguiente muestra cómo conectar una fuente de datos DB2 utilizando IBM OLE DB Provider para DB2:

Ejemplo 1: Aplicación Visual Basic que utiliza ADO

```
Dim db As ADODB.Connection
Set db = New ADODB.Connection
db.Provider = "IBMDADB2"
db.CursorLocation = adUseClient
...
```

Ejemplo 2: Aplicación C/C++ que utiliza IDataInitialize y el componente de servicio

```
hr = CoCreateInstance (
     CLSID_MSDAINITIALIZE,
     NULL,
     CLSCTX_INPROC_SERVER,
     IID_IDataInitialize,
     (void**)&pIDataInitialize);

hr = pIDataInitialize->CreateDBInstance(
     CLSID_IBMDADB2, // ClassID de IBMDADB2
     NULL,
     CLSCTX_INPROC_SERVER,
     NULL,
     IID_IDBInitialize,
     (IUnknown**)&pIDBInitialize);
```

Aplicaciones ADO

Palabras clave de series de conexión de ADO

Para especificar palabras clave de series de conexión de ADO (Objetos de datos ActiveX), especifique la palabra clave con el formato `palabra clave=`*`valor`* en la serie (conexión) del proveedor. Delimite varias palabras clave con un punto y coma (;).

La tabla siguiente describe las palabras clave a las que da soporte IBM OLE DB Provider para DB2:

Tabla 51. Palabras clave soportadas por IBM OLE DB Provider para DB2

Palabra clave	Valor	Significado
DSN	Nombre del alias de la base de datos	El alias de la base de datos DB2 en el directorio de bases de datos.
UID	ID de usuario	El ID de usuario que se utiliza para conectar con el servidor DB2.
PWD	Contraseña de UID	Contraseña correspondiente al ID de usuario utilizado para conectar con el servidor DB2.

Hay otras palabras clave de configuración de CLI de DB2 que también afectan al comportamiento de IBM OLE DB Provider.

Conexiones a fuentes de datos con aplicaciones ADO Visual Basic

Para conectar con una fuente de datos DB2 utilizando IBM OLE DB Provider para DB2, especifique el nombre de proveedor IBMDADB2.

Cursores desplazables actualizables en aplicaciones ADO

IBM OLE DB Provider para DB2 da soporte nativo a cursores de sólo lectura, de sólo avance, desplazables de sólo lectura, así como actualizables y desplazables. Una aplicación ADO que desea acceder a cursores desplazables actualizables puede establecer la ubicación del cursor en `adUseClient` o `adUseServer`. Si establece la ubicación del cursor en `adUseServer`, hace que el cursor se materialice en el servidor.

Limitaciones para aplicaciones ADO

A continuación se describen las limitaciones de las aplicaciones ADO:

- Las aplicaciones ADO que llaman a procedimientos almacenados deben crear y vincular de forma explícita sus parámetros. El método `Parameters.Refresh` para generar parámetros automáticamente no está soportado para DB2 Server para VSE y VM.
- No se da soporte a valores por omisión de parámetros.
- Al insertar una fila nueva utilizando un cursor desplazable en el lado del servidor, utilice el método AddNew() con los argumentos Fieldlist y Values. Esto es más eficaz que invocar AddNew() sin argumentos a continuación de llamadas Update() para cada columna. Cada llamada AddNew() y Update() es una petición independiente para el servidor y por tanto, es menos eficaz que una única llamada a AddNew().

- Las filas recién insertadas no pueden actualizarse con un cursor desplazable en el extremo del servidor.
- Las tablas con datos largos o LOB no se pueden actualizar al utilizar un cursor desplazable en el extremo del servidor.

Soporte de IBM OLE DB Provider para métodos y propiedades ADO

IBM OLE DB Provider da soporte a los siguientes métodos y propiedades ADO:

Tabla 52. Métodos de mandato

Método/Propiedad	Interfaz/Propiedad OLE DB	Soporte de IBM OLE DB
Cancel	ICommand	Sí
CreateParameter		Sí
Execute		Sí

Tabla 53. Propiedades de mandato

Método/Propiedad	Interfaz/Propiedad OLE DB	Soporte de IBM OLE DB
ActiveConnection	(Específico de ADO)	
Command Text	ICommandText	Sí
Command Timeout	ICommandProperties::SetProperties DBPROP_COMMANDTIMEOUT	Sí
CommandType	(Específico de ADO)	
Prepared	ICommandPrepare	Sí
State	(Específico de ADO)	

Tabla 54. Grupos de mandatos

Método/Propiedad	Interfaz/Propiedad OLE DB	Soporte de IBM OLE DB
Parámetros	ICommandWithParameter DBSCHEMA _PROCEDURE_PARAMETERS	Sí
Propiedades	ICommandProperties IDBProperties	Sí

Tabla 55. Métodos de conexión

Método/Propiedad	Interfaz/Propiedad OLE DB	Soporte de IBM OLE DB
BeginTrans CommitTrans RollbackTrans	ITransactionLocal	Sí (pero no anidado) Sí (pero no anidado) Sí (pero no anidado)
Execute	ICommand IOpenRowset	Sí

Tabla 55. Métodos de conexión (continuación)

Método/Propiedad	Interfaz/Propiedad OLE DB	Soporte de IBM OLE DB
Open	IDBCreateSession IDBInitialize	Sí
OpenSchema adSchemaColumnPrivileges adSchemaColumns adSchemaForeignKeys adSchemaIndexes adSchemaPrimaryKeys adSchemaProcedureParam adSchemaProcedures adSchemaProviderType adSchemaStatistics adSchemaTablePrivileges adSchemaTables	IDBSchemaRowset	 Sí Sí Sí Sí Sí Sí Sí Sí Sí Sí Sí
Cancel		Sí

Tabla 56. Propiedades de conexión

Método/Propiedad	Interfaz/Propiedad OLE DB	Soporte de IBM OLE DB
Attributes adXactCommitRetaining adXactRollbackRetaining	ITransactionLocal	 Sí Sí
CommandTimeout	ICommandProperties DBPROP_COMMAND_TIMEOUT	Sí
ConnectionString	(Específico de ADO)	
ConnectionTimeout	IDBProperties DBPROP_INIT_TIMEOUT	No
CursorLocation: adUseClient adUseNone adUseServer	 (Utilizar servicio de cursores de OLE DB) (No utilizado)	 Sí No Sí
DefaultDataBase	IDBProperties DBPROP_CURRENTCATALOG	No
IsolationLevel	ITransactionLocal DBPROP_SESS _AUTOCOMMITISOLEVELS	Sí

Tabla 56. Propiedades de conexión (continuación)

Método/Propiedad	Interfaz/Propiedad OLE DB	Soporte de IBM OLE DB
Mode	IDBProperties	
adModeRead	DBPROP_INIT_MODE	No
adModeReadWrite		Sí
adModeShareDenyNone		No
adModeShareDenyRead		No
adModeShareDenyWrite		No
adModeShareExclusive		No
adModeUnknown		No
adModeWrite		No
Provider	ISourceRowset::GetSourceRowset	Sí
State	(Específico de ADO)	
Version	(Específico de ADO)	

Tabla 57. Grupos de conexiones

Método/Propiedad	Interfaz/Propiedad OLE DB	Soporte de IBM OLE DB
Errors	IErrorRecords	Sí
Propiedades	IDBProperties	Sí

Tabla 58. Propiedades de error

Método/Propiedad	Interfaz/Propiedad OLE DB	Soporte de IBM OLE DB
	IErrorRecords	
Description		Sí
NativeError		Sí
Number		Sí
Source		Sí
SQLState		Sí
HelpContext		No
HelpFile		No

Tabla 59. Métodos de campo

Método/Propiedad	Interfaz/Propiedad OLE DB	Soporte de IBM OLE DB
AppendChunk	ISequentialStream	Sí
GetChunk		Sí

Tabla 60. Propiedades de campo

Método/Propiedad	Interfaz/Propiedad OLE DB	Soporte de IBM OLE DB
Actual Size	IAccessor IRowset	Sí

Tabla 60. Propiedades de campo (continuación)

Método/Propiedad	Interfaz/Propiedad OLE DB	Soporte de IBM OLE DB
Attributes DataFormat DefinedSize Name NumericScale Precision Type	IColumnInfo	 Sí Sí Sí Sí Sí Sí
OriginalValue	IRowsetUpdate	Sí (Servicio de cursores)
UnderlyingValue	IRowsetRefresh IRowsetResynch	Sí (Servicio de cursores) Sí (Servicio de cursores)
Valor	IAccessor IRowset	Sí

Tabla 61. Grupo de campos

Método/Propiedad	Interfaz/Propiedad OLE DB	Soporte de IBM OLE DB
Propiedades	IDBProperties IRowsetInfo	Sí

Tabla 62. Métodos de parámetro

Método/Propiedad	Interfaz/Propiedad OLE DB	Soporte de IBM OLE DB
AppendChunk	ISequentialStream	Sí
Attributes Direction Name NumericScale Precision Scale Size Type	ICommandWithParameter DBSCHEMA _PROCEDURE_PARAMETERS	 Sí No Sí Sí Sí Sí Sí
Valor	IAccessor ICommand	Sí

Tabla 63. Grupo de parámetros

Método/Propiedad	Interfaz/Propiedad OLE DB	Soporte de IBM OLE DB
Propiedades		Sí

Tabla 64. Métodos RecordSet

Método/Propiedad	Interfaz/Propiedad OLE DB	Soporte de IBM OLE DB
AddNew	IRowsetChange	Sí
Cancel		Sí

Tabla 64. Métodos RecordSet (continuación)

Método/Propiedad	Interfaz/Propiedad OLE DB	Soporte de IBM OLE DB
CancelBatch	IRowsetUpdate::Undo	Sí (Servicio de cursores)
CancelUpdate		Sí (Servicio de cursores)
Clone	IRowsetLocate	Sí
Close	IAccessor IRowset	Sí
CompareBookmarks		No
Delete	IRowsetChange	Sí
GetRows	IAccessor IRowset	Sí
Move	IRowset IRowsetLocate	Sí
MoveFirst	IRowset IRowsetLocate	Sí
MoveNext	IRowset IRowsetLocate	Sí
MoveLast	IRowsetLocate	Sí
MovePrevious	IRowsetLocate	Sí
NextRecordSet	IMultipleResults	Sí
Open	ICommand IOpenRowset	Sí
Requery	ICommand IOpenRowset	Sí
Resync	IRowsetRefresh	Sí (Servicio de cursores)
Supports	IRowsetInfo	Sí
Update UpdateBatch	IRowsetChange IRowsetUpdate	Sí Sí (Servicio de cursores)

Tabla 65. Propiedades de RecordSet

Método/Propiedad	Interfaz/Propiedad OLE DB	Soporte de IBM OLE DB
AbsolutePage	IRowsetLocate IRowsetScroll	Sí Sí[1]
AbsolutePosition	IRowsetLocate IRowsetScroll	Sí Sí[1]
ActiveConnection	IDBCreateSession IDBInitialize	Sí

Tabla 65. Propiedades de RecordSet (continuación)

Método/Propiedad	Interfaz/Propiedad OLE DB	Soporte de IBM OLE DB
BOF	(Específico de ADO)	
Bookmark	IAccessor IRowsetLocate	Sí
CacheSize	cRows in IRowsetLocate IRowset	Sí
CursorType adOpenDynamic adOpenForwardOnly adOpenKeySet adOpenStatic	ICommandProperties	 No Sí Sí Sí
EditMode	IRowsetUpdate	Sí (Servicio de cursores)
EOF	(Específico de ADO)	
Filter	IRowsetLocate IRowsetView IRowsetUpdate IViewChapter IViewFilter	No
LockType	ICommandProperties	Sí
MarshallOption		No
MaxRecords	ICommandProperties IOpenRowset	Sí
PageCount	IRowsetScroll	Sí[1]
PageSize	(Específico de ADO)	
Sort	(Específico de ADO)	
Source	(Específico de ADO)	
State	(Específico de ADO)	
Status	IRowsetUpdate	Sí (Servicio de cursores)
Nota: 1. Los valores a devolver son aproximaciones. Las filas suprimidas no se pasarán por alto.		

Tabla 66. Grupo de RecordSet

Método/Propiedad	Interfaz/Propiedad OLE DB	Soporte de IBM OLE DB
Fields	IColumnInfo	Sí
Propiedades	IDBProperties IRowsetInfo::GetProperties	Sí

Compilación y enlace de aplicaciones C/C++ e IBM OLE DB Provider

Las aplicaciones C/C++ que utilizan la constante CLSID_IBMDADB2 deben incluir el archivo `ibmdadb2.h`, que se encuentra en el directorio `SQLLIB\include`. Estas aplicaciones deben definir `DBINITCONSTANTS` antes de la sentencia include. El ejemplo siguiente muestra la secuencia correcta de directrices de preprocesador C/C++:

```
#define DBINITCONSTANTS
#include "ibmdadb2.h"
```

Conexiones con fuentes de datos en aplicaciones C/C++ mediante IBM OLE DB Provider

Para conectar con una fuente de datos DB2 utilizando IBM OLE DB Provider para DB2 en una aplicación C/C++, puede utilizar una de las dos interfaces principales de OLE DB, `IDBPromptInitialize` o `IDataInitialize`, o llamar a la API CoCreateInstance de COM. El componente OLE DB Service expone la interfaz `IDataInitialize` y el componente Data Links expone la interfaz `IDBPromptInitialize`.

Soporte de transacciones distribuidas COM+ e IBM OLE DB Provider

Las aplicaciones OLE DB que se ejecutan en un entorno Microsoft Component Services (COM+) en Windows 2000 o XP pueden utilizar la interfaz `ITransactionJoin` para participar en transacciones distribuidas con varios servidores de bases de datos DB2 Database para Linux, UNIX y Windows, de sistema principal y System i, así como otros gestores de recursos que cumplan con las especificaciones COM+.

Requisitos previos

Para poder utilizar el soporte de transacciones distribuidas COM+ que ofrece IBM OLE DB Provider para DB2, asegúrese de que el servidor cumpla los siguientes requisitos previos.

Nota: Estos requisitos sólo afectan a máquinas basadas en Windows en la que están instalados los clientes DB2.

- Windows 2000 con Service Pack 3 o posterior
- Windows XP

Habilitación del soporte de COM+ en aplicaciones de bases de datos C/C++

Para ejecutar una aplicación C o C++ en modalidad transaccional COM+, puede crear la instancia de fuente de datos IBMDADB2 utilizando `CoCreateInstance`, obtener un objeto de sesión y utilizar `JoinTransaction`. Consulte la descripción sobre cómo conectar una aplicación C o C++ a una fuente de datos para obtener más información.

Para ejecutar una aplicación ADO en modalidad transaccional COM+, consulte la descripción sobre cómo conectar una aplicación C o C++ a una fuente de datos.

Para utilizar un componente de un paquete COM+ en modalidad transaccional, establezca para la propiedad Transactions del componente uno de los siguientes valores:

- “Required”
- “Required New”
- “Supported”

Para obtener información sobre estos valores, consulte la documentación de COM+.

Capítulo 6. OLE DB .NET Data Provider

OLE DB .NET Data Provider utiliza el Controlador IBM DB2 OLE DB, al que se hace referencia en un objeto `ConnectionString` como `IBMDADB2`. Las palabras clave de la serie de conexión soportadas por OLE DB .NET Data Provider son las misma que las soportadas por IBM OLE DB Provider para DB2. Además, OLE DB .NET Data Provider tiene las mismas restricciones que IBM DB2 OLE DB Provider. Existen restricciones adicionales para OLE DB .NET Data Provider, que se describen en el tema: Restricciones de OLE DB .NET Data Provider.

Para utilizar OLE DB .NET Data Provider, debe tener .NET Framework versión 1.1, 2.0 o 3.0 instalada.

Para DB2 Universal Database para AS/400 e iSeries, es necesario el siguiente arreglo de programa en el servidor: APAR ii13348.

Estas son las palabras clave de conexión soportadas para OLE DB .NET Data Provider:

Tabla 67. Palabras clave útiles de ConnectionString para OLE DB .NET Data Provider

Palabra clave	Valor	Significado
`PROVIDER`	`IBMDADB2`	Especifica el IBM OLE DB Provider para DB2 (obligatorio)
`DSN` o `Data Source`	alias de base de datos	El alias de la base de datos DB2 tal como está catalogado en el directorio de bases de datos.
`UID`	ID de usuario	ID de usuario que se utiliza para conectar con el servidor de base de datos DB2.
`PWD`	contraseña	La contraseña correspondiente al ID de usuario utilizado para conectar con el servidor DB2

Nota: Para obtener la lista completa de las palabras clave ConnectionString, consulte la documentación de Microsoft.

El ejemplo siguiente crea una conexión `OleDbConnection` para conectar con la base de datos SAMPLE:

```
[Visual Basic .NET]
Dim con As New OleDbConnection("Provider=IBMDADB2;" +
    "Data Source=sample;UID=userid;PWD=password;")
con.Open()

[C#]
OleDbConnection con = new OleDbConnection("Provider=IBMDADB2;" +
                "Data Source=sample;UID=userid;PWD=password;" );
con.Open()
```

Restricciones de OLE DB .NET Data Provider

La tabla siguiente muestra las restricciones de utilización de IBM OLE DB .NET Data Provider:

Tabla 68. Restricciones de IBM OLE DB .NET Data Provider

Clase o función	Descripción de la restricción	Servidores DB2 afectados
Corrientes de caracteres ASCII	No puede utilizar corrientes de caracteres ASCII con `OleDbParameters` si utiliza `DbType.AnsiString` o `DbType.AnsiStringFixedLength`. OLE DB .NET Data Provider emitirá la excepción siguiente: `"La conversión de tipo de datos especificada no es válida"`	Todos
`ADORecord`	`ADORecord` no está soportado.	Todos
`ADORecordSet` y `Timestamp`	Tal como está documentado en MSDN, la variable `ADORecordSet` toma el valor 1 segundo. Cono consecuencia, todas las fracciones de segundo se pierden cuando una columna `Timestamp` de DB2 se almacena en un `ADORecordSet`. Similarmente, después de llenar un `DataSet` de un `ADORecordSet`, las columnas `Timestamp` de `DataSet` no tendrán fracciones de segundo.	Todos
`Capítulos`	Los capítulos no están soportados.	Todos
Información de claves	OLE DB .NET Data Provider no puede obtener información de claves al abrir un `IDataReader` al mismo tiempo.	DB2 para VM/VSE
Información de claves procedente de procedimientos almacenados	OLE DB .NET Data Provider puede recuperar información de claves referente a un conjunto de resultados devuelto por un procedimiento almacenado solo desde DB2 Database para Linux, UNIX y Windows. Esto es debido a que los servidores DB2 para plataformas que no sean Linux, UNIX y Windows no devuelven información descriptiva ampliada para los conjuntos de resultados abiertos en el procedimiento almacenado. Para recuperar información de claves de un conjunto de resultados devuelto por un procedimiento almacenado en DB2 Database para Linux, UNIX y Windows, es necesario definir la siguiente variable de registro en el servidor DB2: `db2set DB2_APM_PERFORMANCE=8` El definir esta variable de registro de DB2 del servidor hará que los metadatos del conjunto de resultados estén disponibles en el servidor durante más tiempo, lo cual permite que OLE DB recupere satisfactoriamente la información de claves. Sin embargo, dependiendo de la carga de trabajo del servidor, puede que los metadatos no estén disponibles el tiempo suficiente para que OLE DB Provider consulte la información. Por tanto, no es seguro que la información de claves esté siempre disponible para los conjuntos de resultados devueltos por un procedimiento almacenado. Para recuperar cualquier información de claves sobre una sentencia CALL, la aplicación debe ejecutar la sentencia CALL. La invocación de `OleDbDataAdapter.FillSchema()` o `OleDbCommand.ExecuteReader(CommandBehavior.SchemaOnly | CommandBehavior.KeyInfo)` no ejecutará realmente la llamada de procedimiento almacenado. Por tanto, el usuario no obtiene ninguna información de claves para el conjunto de resultados que debe ser devuelto por el procedimiento almacenado.	Todos

Tabla 68. Restricciones de IBM OLE DB .NET Data Provider (continuación)

Clase o función	Descripción de la restricción	Servidores DB2 afectados

Información de claves de sentencias de SQL de proceso por lotes	Cuando se utilizan sentencias de SQL de proceso por lotes que devuelven varios resultados, el método `FillSchema()` intenta recuperar información de esquemas solo para la primera sentencia de SQL contenida en la lista de sentencias de SQL de proceso por lotes. Si esta sentencia no devuelve un conjunto de resultados, no se crea ninguna tabla. Por ejemplo:
	`[C#]`
	`cmd.CommandText = "INSERT INTO ORG(C1) VALUES(1000); SELECT C1 FROM ORG;";`
	`da = new OleDbDataAdapter(cmd);`
	`da.FillSchema(ds, SchemaType.Source);`
	
	No se creará ninguna tabla en el archivo, pues la primera sentencia de SQL del lote es una sentencia INSERT, que no devuelve un conjunto de resultados.
		Todos

`OleDbCommandBuilder`	Las sentencias UPDATE, DELETE e INSERT generadas automáticamente por `OleDbCommandBuilder` son incorrectas si la sentencia SELECT contiene cualquier columna de los tipos de datos siguientes: • CLOB • BLOB • DBCLOB • LONG VARCHAR • LONG VARCHAR FOR BIT DATA • LONG VARGRAPHIC Si se conecta a un servidor DB2 que no sea DB2 Database para Linux, UNIX y Windows, este problema también es debido a columnas con los tipos de datos siguientes: • VARCHAR[1] • VARCHAR FOR BIT DATA[1] • VARGRAPHIC[1] • REAL • FLOAT o DOUBLE • TIMESTAMP **Nota:** 1. Las columnas con estos tipos de datos son aplicables si están definidas para ser valores VARCHAR mayores que 254 bytes, valores VARCHAR FOR BIT DATA mayores que 254 bytes o valores VARGRAPHICs mayores que 127 bytes. Esta condición solo es válida si se conecta a un servidor DB2 que no sea DB2 Database para Linux, UNIX y Windows. `OleDbCommandBuilder` genera sentencias de SQL que utilizan todas las columnas seleccionadas en una comparación de igualdad de la cláusula WHERE, pero los tipos de datos listados anteriormente no se pueden utilizar en una comparación de igualdad. **Nota:** Observe que esta restricción no afectará al método `IDbDataAdapter.Update()` que depende de `OleDbCommandBuilder` para generar automáticamente las sentencias UPDATE, DELETE e INSERT. La operación UPDATE no es efectiva si la sentencia generada contiene cualquiera de los tipos de datos listados anteriormente.	Todos
`OleDbCommandBuilder.DeriveParameters`	La distinción entre mayúsculas y minúsculas es importante al utilizar `DeriveParameters()`. El nombre de procedimiento almacenado especificado en `OleDbCommand.CommandText` necesita estar escrito exactamente tal como está almacenado en las tablas de catálogo del sistema DB2. Para ver cómo están almacenados los nombres de procedimiento almacenado, ejecute OpenSchema(OleDbSchemaGuid.Procedures) sin proporcionar la restricción de nombre del procedimiento. Esto devolverá todos los nombres de procedimiento almacenado. Por omisión, DB2 almacena los nombres de procedimiento almacenado en letras mayúsculas, por lo que generalmente deberá especificar el nombre del procedimiento almacenado en mayúsculas.	Todos

Tabla 68. Restricciones de IBM OLE DB .NET Data Provider (continuación)

Clase o función	Descripción de la restricción	Servidores DB2 afectados
`OleDbCommandBuilder. DeriveParameters`	El método `OleDbCommandBuilder.DeriveParameters()` no incluye el parámetro `ReturnValue` en el objeto `OleDbParameterCollection` generado. SqlClient y IBM Data Server Provider para .NET añaden por omisión el parámetro con `ParameterDirection.ReturnValue` al objeto `ParameterCollection` generado.	Todos
`OleDbCommandBuilder. DeriveParameters`	El método `OleDbCommandBuilder.DeriveParameters()` no es efectivo para procedimientos almacenados sobrecargados. Si existen varios procedimientos almacenados denominados "MYPROC", cada uno de los cuales acepta un número diferente de parámetros o tipos diferentes de parámetros, `OleDbCommandBuilder.DeriveParameters()` recupera todos los parámetros de todos los procedimientos almacenados sobrecargados.	Todos
`OleDbCommandBuilder. DeriveParameters`	Si la aplicación no autoriza un procedimiento almacenado con un esquema, `DeriveParameters()` devuelve todos los parámetros para ese nombre de procedimiento. Por tanto, si existen varios esquemas para el mismo nombre de procedimiento, `DeriveParameters()` devuelve todos los parámetros para todos los procedimientos del mismo nombre.	Todos
`OleDbConnection. ChangeDatabase`	El método `OleDbConnection.ChangeDatabase()` no está soportado.	Todos
`OleDbConnection. ConnectionString`	El uso de caracteres no imprimibles, tales como '\b', '\a' o '\O' en la serie de conexión hará que se emita una excepción. Existen restricciones para las palabras clave siguientes: **`Fuente de datos`** La fuente de datos es el nombre de la base de datos, no el servidor. Puede especificar la palabra clave SERVER, pero no será tenida en cuenta por el proveedor IBMDADB2. **`Initial Catalog` y `Connect Timeout`** Estas palabras clave no están soportadas. En general, OLE DB .NET Data Provider no tiene en cuenta ninguna de las palabras clave no reconocidas y no soportadas. Pero si especifica estas palabras clave, se emite la excepción siguiente: `La operación de OLE DB de varios pasos ha generado errores. Examine cada valor de estado de OLE DB, si está disponible. No se ha realizado ninguna acción.` **`ConnectionTimeout`** `ConnectionTimeout` es de solo lectura.	Todos

`OleDbConnection. GetOleDbSchemaTable`	Los valores de restricción distinguen entre mayúsculas y minúsculas, y necesitan estar escritos exactamente tal como están especificados los objetos de base de datos almacenados en las tablas de catálogo del sistema. Por omisión, esos valores se almacenan en mayúsculas.
	
	Por ejemplo, si ha creado una tabla de la manera siguiente:
	`CREATE TABLE abc(c1 SMALLINT)`
	
	DB2 almacenará el nombre de tabla en mayúsculas ("ABC") en el catálogo del sistema. Por tanto, debe utilizar "ABC" como valor de restricción. Por ejemplo:
	`schemaTable = con.GetOleDbSchemaTable(OleDbSchemaGuid.Tables, new object[] { null, null, "ABC", "TABLE" });`
		Todos

`OleDbDataAdapter` y `DataColumnMapping`	El nombre de la columna fuente distingue entre mayúsculas y minúsculas. Debe estar escrita exactamente tal como está almacenada en los catálogos de DB2, que, por omisión, es en mayúsculas. Por ejemplo: `colMap = new DataColumnMapping("EMPNO", "Employee ID");`	Todos

Tabla 68. Restricciones de IBM OLE DB .NET Data Provider (continuación)

Clase o función	Descripción de la restricción	Servidores DB2 afectados
`OleDbDataReader.GetSchemaTable`	OLE DB .NET Data Provider no puede recuperar información descriptiva ampliada procedente de servidores que no devuelven esa clase de información. Si se conecta a un servidor que no da soporte a información descriptiva ampliada (los servidores afectados), las columnas siguientes de la tabla de metadatos devuelta desde `IDataReader.GetSchemaTable()` no son válidas: • `IsReadOnly` • `IsUnique` • `IsAutoIncrement` • `BaseSchemaName` • `BaseCatalogName`	DB2 para OS/390, versión 7 o inferior DB2 para OS/400 DB2 para VM/VSE
Procedimientos almacenados: sin nombres de columna para conjuntos de resultados	El servidor DB2 para OS/390 versión 6.1 no devuelve nombres de columna para conjuntos de resultados devueltos desde un procedimiento almacenado. OLE DB .NET Data Provider correlaciona estas columnas sin nombre con su número de posición ordinal (por ejemplo, ″1″, ″2″ ″3″). Esto es diferente de la correlación documentada en MSDN: `"Columna1", "Columna2", "Columna3"`.	DB2 para OS/390 versión 6.1

Sugerencias y consejos

Agrupación de conexiones en aplicaciones de OLE DB .NET Data Provider

OLE DB .NET Data Provider agrupa automáticamente las conexiones utilizando el servicio de agrupación de sesiones de OLE DB. Se pueden utilizar argumentos de la serie de conexión para habilitar o inhabilitar servicios de OLE DB, incluida la agrupación de conexiones. Por ejemplo, la siguiente serie de conexión inhabilita la agrupación de sesiones y el enlistamiento automático de transacciones de OLE DB.

```
Provider=IBMDADB2;OLE DB Services=-4;Data Source=SAMPLE;
```

La tabla siguiente describe los atributos de la serie de conexión de ADO que puede utilizar para definir los servicios de OLE DB:

Tabla 69. Definición de servicios de OLE DB mediante atributos de la serie de conexión de ADO

Servicios habilitados	Valor en la serie de conexión
Todos los servicios (valor por omisión)	″OLE DB Services = -1;″
Todos los servicios excepto la agrupación	″OLE DB Services = -2;″
Todos los servicios excepto la agrupación y el enlistamiento automático	″OLE DB Services = -4;″
Todos los servicios excepto el cursor de cliente	″OLE DB Services = -5;″
Todos los servicios excepto el cursor de cliente y la agrupación	″OLE DB Services = -6;″
Ningún servicio	″OLE DB Services = 0;″

Para obtener más información sobre la agrupación de sesiones o de recursos de OLE DB, así como sobre la inhabilitación del servicio de agrupación mediante la modificación de los valores por omisión del servicio proveedor de OLE DB, consulte el manual OLE DB Programmer's Reference que se encuentra en la biblioteca de MSDN situada en:

http://msdn.microsoft.com/library

Columnas de tiempo en aplicaciones de OLE DB .NET Data Provider

Las secciones siguientes describen implementar columnas de tiempo en aplicaciones de OLE DB .NET Data Provider.

Inserción marcadores de parámetros

Por ejemplo, suponga que desea insertar un valor de tiempo en una columna de Tiempo:

```
command.CommandText = "insert into mytable(c1) values( ? )";
```

donde la columna c1 es una columna de Tiempo. He aquí dos métodos para vincular un valor de tiempo al marcador de parámetros:

Utilizando `OleDbParameter.OleDbType = OleDbType.DBTime`

Debido a que OleDbType.DBTime se correlaciona con un objeto TimeSpan, debe proporcionar un objeto TimeSpan como valor de parámetro. El valor de parámetro no puede ser un objeto String ni DateTime; debe ser un objeto TimeSpan. Por ejemplo:

```
p1.OleDbType = OleDbType.DBTime;
p1.Value = TimeSpan.Parse("0.11:20:30");
rowsAffected = cmd.ExecuteNonQuery();
```

El formato de TimeSpan es una serie de caracteres de la forma: ″[-]d.hh:mm:ss.ff″, tal como está documentado en el documentación de MSDN.

Utilizando `OleDbParameter.OleDbType = OleDbType.DateTime`

Esta especificación hará que OLE DB .NET Data Provider convierta el valor de parámetro en un objeto DateTime, en lugar de un objeto TimeSpan. Como consecuencia, el valor de parámetro puede ser cualquier serie/objeto válido que se pueda convertir en un objeto DateTime. Esto significa que valores tales como ″11:20:30″ serán efectivos. El valor también puede ser un objeto DateTime. El valor no puede ser un objeto TimeSpan, pues no se puede convertir en un objeto DateTime. TimeSpan no implementa IConvertible.

Por ejemplo:

```
p1.OleDbType = OleDbType.DBTimeStamp;
p1.Value = "11:20:30";
rowsAffected = cmd.ExecuteNonQuery();
```

Recuperación

Para recuperar una columna de tiempo necesita utilizar el método `IDataRecord.GetValue()` o `OleDbDataReader.GetTimeSpan()`.

Por ejemplo:

```
TimeSpan ts1 = ((OleDbDataReader)reader).GetTimeSpan( 0 );
TimeSpan ts2 = (TimeSpan) reader.GetValue( 0 );
```

Objetos ADORecordset en aplicaciones de OLE DB .NET Data Provider

Las consideraciones siguientes tratan sobre la utilización de objetos ADORecordset.

- La clase adDBTime de tipo ADO se correlaciona con la clase DateTime de .NET Framework. OleDbType.DBTime corresponde a un objeto TimeSpan.
- No puede asignar un objeto TimeSpan al campo Time de un objeto ADORecordset. Esto es debido a que el campo Time del objeto ADORecordset espera recibir un objeto DateTime. Cuando asigna un objeto TimeSpan a un objeto ADORecordset, recibe el mensaje siguiente:

  ```
  La signatura de tipo del método no es compatible con Interop.
  ```

  Solo puede llenar el campo Time con un objeto DateTime, o con un objeto String que se pueda convertir en un objeto DateTime.
- Cuando llena un DataSet con un ADORecordset utilizando OleDbDataAdapter, el campo Time de ADORecordset se convierte en una columna TimeSpan de DataSet.
- Los Recordsets no almacenan claves primarias ni restricciones. Por tanto, no se añade ninguna información de claves al llenar un DataSet a partir de un Recordset utilizando MissingSchemaAction.AddWithKey.

Capítulo 7. ODBC .NET Data Provider

ODBC .NET Data Provider realiza llamadas de ODBC a una fuente de datos DB2 utilizando el controlador de CLI de DB2. Por tanto, las palabras de serie de conexión soportadas por ODBC .NET Data Provider son las mismas que las soportadas por el controlador de CLI de DB2. Además, ODBC DB .NET Data Provider tiene las mismas restricciones que el controlador de CLI de DB2. Existen restricciones adicionales para ODBC .NET Data Provider, que se describen en el tema: Restricciones de ODBC .NET Data Provider.

Para utilizar ODBC .NET Data Provider, debe tener .NET Framework versión 1.1, 2.0 o 3.0 instalado. Para DB2 Universal Database para AS/400 e iSeries, es necesario el siguiente arreglo para el servidor: APAR II13348.

Estas son las palabras clave de conexión soportadas para ODBC .NET Data Provider:

Tabla 70. Palabras clave útiles de ConnectionString para ODBC .NET Data Provider

Palabra clave	Valor	Significado
`DSN`	alias de base de datos	El alias de la base de datos DB2 tal como está catalogado en el directorio de bases de datos.
`UID`	ID de usuario	ID de usuario que se utiliza para conectar con el servidor DB2.
`PWD`	contraseña	La contraseña correspondiente al ID de usuario utilizado para conectar con el servidor DB2

Nota: Para obtener la lista completa de las palabras clave ConnectionString, consulte la documentación de Microsoft.

El ejemplo siguiente crea una conexión `OdbcConnection` para conectar con la base de datos SAMPLE:

```
[Visual Basic .NET]
Dim con As New OdbcConnection("DSN=sample;UID=userid;PWD=password;")
con.Open()

[C#]
OdbcConnection con = new OdbcConnection("DSN=sample;UID=userid;PWD=password;");
con.Open()
```

Restricciones de ODBC .NET Data Provider

La tabla siguiente muestra las restricciones de utilización de IBM ODBC .NET Data Provider:

Tabla 71. Restricciones de IBM ODBC .NET Data Provider

Clase o función	Descripción de la restricción	Servidores DB2 afectados
Corrientes de caracteres ASCII	No puede utilizar corrientes de caracteres ASCII con `OdbcParameters` si utiliza `DbType.AnsiString` o `DbType.AnsiStringFixedLength`. ODBC .NET Data Provider emitirá la excepción siguiente: `"La conversión de tipo de datos especificada no es válida"`	Todos

`Command.Prepare`	Antes de ejecutar un mandato (`Command.ExecuteNonQuery` o `Command.ExecuteReader`), debe ejecutar explícitamente `OdbcCommand.Prepare()` si `CommandText` ha cambiado desde la última preparación. Si no ejecuta `OdbcCommand.Prepare()` de nuevo, ODBC .NET Data Provider ejecutará el `CommandText` preparado anteriormente.
	
	Por ejemplo:
	`[C#]`
	`command.CommandText="select CLOB('ABC') from table1";`
	`command.Prepare();`
	`command.ExecuteReader();`
	`command.CommandText="select CLOB('XYZ') from table2";`
	
	`// Esto finaliza la repetición de la ejecución de la primera sentencia`
	`command.ExecuteReader();`
		Todos

CommandBehavior. SequentialAccess	Si utiliza `IDataReader.GetChars()` para leer con un lector creado con `CommandBehavior.SequentialAccess`, debe asignar un almacenamiento intermedio que sea lo suficiente grande para contener la columna completa. De lo contrario, recibirá la excepción siguiente:
	`El rango solicitado se extiende más allá del final de la matriz.`
	`at System.Runtime.InteropServices.Marshal.Copy(Int32 source,`
	`Char[] destination, Int32 startIndex, Int32 length)`
	`at System.Data.Odbc.OdbcDataReader.GetChars(Int32 i,`
	`Int64 dataIndex, Char[] buffer, Int32 bufferIndex, Int32 length)`
	`at OleRestrict.TestGetCharsAndBufferSize(IDbConnection con)`
	
	El ejemplo siguiente muestra cómo asignar un almacenamiento intermedio apropiado:
	`CREATE TABLE myTable(c0 int, c1 CLOB(10K))`
	`SELECT c1 FROM myTable;`
	
	`[C#]`
	`cmd.CommandText = "SELECT c1 from myTable";`
	`IDataReader reader = cmd.ExecuteReader(CommandBehavior.SequentialAccess);`
	
	`Int32 iChunkSize = 10;`
	`Int32 iBufferSize = 10;`
	`Int32 iFieldOffset = 0;`
	
	`Char[] buffer = new Char[ iBufferSize ];`
	
	`reader.Read();`
	`reader.GetChars(0, iFieldOffset, buffer, 0, iChunkSize);`
	
	La llamada a `GetChars()` emitirá la excepción siguiente:
	`"El rango solicitado se extiende más allá del final de la matriz"`
	
	Para evitar que `GetChars()` emita la excepción anterior, asigne a `BufferSize` un valor igual al tamaño de la columna, de esta manera:
	`Int32 iBufferSize = 10000;`
	
	Observe que el valor de 10000 para `iBufferSize` corresponde al valor de 10K asignado a la columna CLOB `c1`.
		Todos

Tabla 71. Restricciones de IBM ODBC .NET Data Provider (continuación)

Clase o función	Descripción de la restricción	Servidores DB2 afectados

CommandBehavior. SequentialAccess	ODBC .NET Data Provider emite la excepción siguiente cuando no existen más datos para leer al utilizar `OdbcDataReader.GetChars()`:
	`NO_DATA - No hay disponible información del error`
	`at System.Data.Odbc.OdbcConnection.HandleError(`
	`HandleRef hrHandle, SQL_HANDLE hType, RETCODE retcode)`
	`at System.Data.Odbc.OdbcDataReader.GetData(`
	`Int32 i, SQL_C sqlctype, Int32 cb)`
	`at System.Data.Odbc.OdbcDataReader.GetChars(`
	`Int32 i, Int64 dataIndex, Char[] buffer,`
	`Int32 bufferIndex, Int32 length)`
		Todos

CommandBehavior. SequentialAccess	No podrá utilizar tamaños de bloques de datos grandes, tales como el representado por el valor 5000, al utilizar `OdbcDataReader.GetChars()`. Cuando intente utilizar un tamaño de bloque de datos grande, ODBC .NET Data Provider emitirá esta excepción:
	`Referencia del objeto no establecida en una instancia de un objeto`
	`at System.Runtime.InteropServices.Marshal.Copy(Int32 source,`
	`Char[] destination, Int32 startIndex, Int32 length)`
	`at System.Data.Odbc.OdbcDataReader.GetChars(`
	`Int32 i, Int64 dataIndex, Char[] buffer,`
	`Int32 bufferIndex, Int32 length)`
	`at OleRestrict.TestGetCharsAndBufferSize(IDbConnection con)`
		Todos

Agrupación de conexiones	ODBC .NET Data Provider no controla la agrupación de conexiones. La agrupación de conexiones es gestionada por el Gestor de controladores ODBC. Para obtener más información sobre la agrupación de conexiones, consulte el manual ODBC Programmer's Reference de la biblioteca de MSDN situada en `http://msdn.microsoft.com/library`	Todos
DataColumnMapping	El nombre de la columna fuente debe coincidir exactamente con el nombre utilizado en las tablas de catálogo del sistema, que por omisión se escribe en mayúsculas.	Todos

Columnas decimales	No se pueden utilizar marcadores de parámetros para las columnas decimales.
	
	Generalmente utilizará `OdbcType.Decimal` para un `OdbcParameter` si el `SQLType` de destino es una columna `Decimal`; sin embargo, cuando ODBC .NET Data Provider lee el `OdbcType.Decimal`, vincula el parámetro utilizando el tipo C de `SQL_C_WCHAR` y `SQLType` de `SQL_VARCHAR`, lo cual no es válido.
	
	Por ejemplo:
	`[C#]`
	`cmd.CommandText = "SELECT dec_col FROM MYTABLE WHERE dec_col > ? ";`
	`OdbcParameter p1 = cmd.CreateParameter();`
	`p1.DbType = DbType.Decimal;`
	`p1.Value = 10.0;`
	`cmd.Parameters.Add(p1);`
	`IDataReader rdr = cmd.ExecuteReader();`
	
	Obtendrá una excepción:
	`ERROR [07006] [IBM][CLI Driver][SQLDS/VM]`
	`SQL0301N El valor de la variable o parámetro de sistema principal`
	`de entrada`
	`number "" no se puede utilizar porque es de tipo de datos`
	`SQLSTATE=07006`
		DB2 para VM/VSE

Tabla 71. Restricciones de IBM ODBC .NET Data Provider (continuación)

Clase o función	Descripción de la restricción	Servidores DB2 afectados

Información de claves	El nombre de esquema utilizado para calificar el nombre de tabla (por ejemplo, `MYSCHEMA.MYTABLE`) debe coincidir con el ID de usuario de la conexión. ODBC .NET Data Provider no puede recuperar ninguna información de claves en la que el esquema especificado sea diferente del ID de usuario de la conexión.
	
	Por ejemplo:
	`CREATE TABLE USERID2.TABLE1(c1 INT NOT NULL PRIMARY KEY);`
	
	`[C#]`
	`// Conectar como usuario bob`
	`odbcCon = new OdbcConnection("DSN=sample;UID=bob;PWD=mypassword");`
	
	`OdbcCommand cmd = odbcCon.CreateCommand();`
	
	`// Seleccionar de la tabla con esquema USERID2`
	`cmd.CommandText="SELECT * FROM USERID2.TABLE1";`
	
	`// Error - no se recupera información de claves`
	`da.FillSchema(ds, SchemaType.Source);`
	
	`// Error - SchemaTable no tiene ninguna clave primaria`
	`cmd.ExecuteReader(CommandBehavior.KeyInfo)`
	
	`// Emite una excepción debido a la falta de claves primarias`
	`cbuilder.GetUpdateCommand();`
		Todos

Información de claves	ODBC .NET Data Provider no puede obtener información de claves al abrir un `IDataReader` al mismo tiempo. Cuando ODBC .NET Data Provider abre un `IDataReader`, se abre un cursor en el servidor. Si se solicita información de claves, se invocará `SQLPrimaryKeys()` o `SQLStatistic()` para obtener la información de claves, pero estas funciones de esquema abrirán otro cursor. Debido a que DB2 para VM/VSE no da soporte a la retención de cursor, se cierra el primer cursor. Como consecuencia, las llamadas de `IDataReader.Read()` al `IDataReader` producen la excepción siguiente:
	`System.Data.Odbc.OdbcException: ERROR [HY010] [IBM][CLI Driver]`
	`CLI0125E Error de secuencia de funciones. SQLSTATE=HY010`
		DB2 para VM/VSE

Información de claves	Los nombres utilizados en sus sentencias de SQL para especificar objetos de base de datos deben coincidir exactamente con los nombres utilizados para esos objetos en las tablas de catálogo del sistema. Por omisión, los nombres de los objetos de base de datos se guardan en mayúsculas en las tablas de catálogo del sistema. Generalmente, pues, deberá utilizar letras mayúsculas. ODBC .NET Data Provider examina sentencias de SQL para recuperar nombres de objetos de base de datos y los pasa a funciones de esquema, tales como `SQLPrimaryKeys` y `SQLStatistics`, que emiten consultas para estos objetos en las tablas de catálogo del sistema. Los nombres utilizados para especificar los objetos de base de datos deben coincidir exactamente con los nombres correspondientes que están almacenados en las tablas de catálogo del sistema; de lo contrario se devuelve un conjunto de resultados vacío.	DB2 para OS/390 DB2 para OS/400 DB2 para VM/VSE
Información de claves para sentencias de SQL de proceso por lotes distintas de SELECT	ODBC .NET Data Provider no puede recuperar ninguna información de claves para una sentencia de proceso por lotes que no comience por ″SELECT″.	DB2 para OS/390 DB2 para OS/400 DB2 para VM/VSE

Tabla 71. Restricciones de IBM ODBC .NET Data Provider (continuación)

Clase o función	Descripción de la restricción	Servidores DB2 afectados

Columnas LOB	ODBC .NET Data Provider no da soporte a los tipos de datos LOB. Como consecuencia, cada vez que el servidor DB2 devuelva un SQL_CLOB (-99), SQL_BLOB (-98) o SQL_DBCLOB (-350), ODBC .NET Data Provider emitirá la excepción siguiente:
	`"Unknown SQL type - -98"  (para una columna Blob)`
	`"Unknown SQL type - -99"  (para una columna Clob)`
	`"Unknown SQL type - -350"  (para una columna DbClob)`
	
	Cualquier método que acceda directa o indirectamente a columnas LOB fallará.
		Todos

`OdbcCommand.Cancel`	La ejecución de sentencias después de ejecutar `OdbcCommand.Cancel` puede producir la excepción siguiente: `"ERROR [24000] [Microsoft][ODBC Driver Manager] Invalid cursor state"`	Todos

`OdbcCommandBuilder`	`OdbcCommandBuilder` no puede generar mandatos para servidores que no dan soporte a caracteres de escape. Cuando `OdbcCommandBuilder` genera mandatos, primero realiza una llamada a `SQLGetInfo`, solicitando el atributo `SQL_SEARCH_PATTERN_ESCAPE`. Si el servidor no soporta caracteres de escape, se devolverá una serie vacía, lo que hace que ODBC .NET Data Provider emita la siguiente excepción:
	`El índice estaba fuera de los límites de la matriz.`
	`at System.Data.Odbc.OdbcConnection.get_EscapeChar()`
	`at System.Data.Odbc.OdbcDataReader.GetTableNameFromCommandText()`
	`at System.Data.Odbc.OdbcDataReader.BuildMetaDataInfo()`
	`at System.Data.Odbc.OdbcDataReader.GetSchemaTable()`
	`at System.Data.Common.CommandBuilder.BuildCache(Boolean closeConnection)`
	`at System.Data.Odbc.OdbcCommandBuilder.GetUpdateCommand()`
		DB2 para OS/390, sólo servidores DBCS; DB2 para VM/VSE, sólo sevidores DBCS

`OdbcCommandBuilder`	La distinción entre letras mayúsculas y minúsculas es importante cuando se utiliza `OdbcCommandBuilder` para generar automáticamente sentencias UPDATE, DELETE e INSERT. Por omisión, DB2 almacena la información de esquema (como nombres de tabla y nombres de columna) en letras mayúsculas en las tablas de catálogo del sistema, a menos que se hayan creado explícitamente con discriminación de mayúsculas y minúsculas (encerrando entre comillas los nombres de objetos de base de datos en el momento de su creación). Las sentencias de SQL que defina deben utilizar nombres escritos exactamente tal como están almacenados en los catálogos (por omisión, los nombres se almacenan en mayúsculas).
	
	Por ejemplo, si ha creado una tabla utilizando la sentencia siguiente:
	`"db2 create table mytable (c1 int) "`
	
	DB2 almacenará el nombre de tabla ″mytable″ como ″MYTABLE″ en las tablas de catálogo del sistema.
	
	El ejemplo de código siguiente muestra la utilización apropiada de la clase `OdbcCommandBuilder`:
	`[C#]`
	`OdbcCommand cmd = odbcCon.CreateCommand();`
	`cmd.CommandText  = "SELECT * FROM MYTABLE";`
	`OdbcDataAdapter da = new OdbcDataAdapter(cmd);`
	`OdbcCommandBuilder cb = new OdbcCommandBuilder(da);`
	`OdbcCommand updateCmd = cb.GetUpdateCommand();`
	
	En este ejemplo, si no especifica el nombre de tabla en mayúsculas, obtendrá la excepción siguiente:
	`"La generación de SQL dinámico para UpdateCommand no está soportada para un SelectCommand que no devuelve ninguna información de columnas de clave."`
		Todos

Tabla 71. Restricciones de IBM ODBC .NET Data Provider (continuación)

Clase o función	Descripción de la restricción	Servidores DB2 afectados

OdbcCommandBuilder	Los mandatos generados por `OdbcCommandBuilder` son incorrectos cuando la sentencia SELECT contiene los tipos de datos de columna siguientes:
	`REAL`
	`FLOAT o DOUBLE`
	`TIMESTAMP`
	
	Estos tipos de datos no se pueden utilizar en la cláusula WHERE de sentencias SELECT.
		DB2 para OS/390
		DB2 para OS/400
		DB2 para VM/VSE

OdbcCommandBuilder. DeriveParameters	El método `DeriveParameters()` se correlaciona con `SQLProcedureColumns` y utiliza la propiedad `CommandText` para el nombre del procedimiento almacenado. Debido a que `CommandText` no contiene el nombre del procedimiento almacenado (utilizando la sintaxis completa de la llamada de ODBC), `SQLProcedureColumns` se invoca con el nombre de procedimiento identificado de acuerdo con la sintaxis de la llamada de ODBC. Por ejemplo: `"{ CALL myProc(?) }"` Esto producirá un conjunto de resultados vacío, cuando no se encuentren columnas para el procedimiento.	Todos
OdbcCommandBuilder. DeriveParameters	Para utilizar `DeriveParameters()`, especifique el nombre de procedimiento almacenado en `CommandText` (por ejemplo, `cmd.CommandText = "MYPROC"`). El nombre del procedimiento almacenado debe estar escrito exactamente tal como está almacenado en las tablas de catálogo del sistema. `DeriveParameters()` devolverá todos los parámetros para ese nombre de procedimiento que encuentre en las tablas de catálogo del sistema. Recuerde que debe restaurar la sintaxis completa de llamada de ODBC para `CommandText` antes de ejecutar la sentencia.	Todos
OdbcCommandBuilder. DeriveParameters	El parámetro `ReturnValue` no se devuelve para ODBC .NET Data Provider.	Todos
OdbcCommandBuilder. DeriveParameters	`DeriveParameters()` no da soporte a nombres de procedimiento almacenado totalmente calificados. Por ejemplo, la invocación de `DeriveParameters()` para `CommandText = "MYSCHEMA.MYPROC"` fallará. En este caso, no se devuelve ningún parámetro.	Todos
OdbcCommandBuilder. DeriveParameters	`DeriveParameters()` no es efectivo para procedimientos almacenados sobrecargados. `SQLProcedureColumns` devolverá todos los parámetros para todas las versiones del procedimiento almacenado.	Todos
`OdbcConnection. ChangeDatabase`	El método `OdbcConnection.ChangeDatabase()` no está soportado.	Todos
`OdbcConnection. ConnectionString`	• La palabra clave `Server` no se tiene en cuenta. • La palabra clave `Connect Timeout` no se tiene en cuenta. La CLI de DB2 no da soporte a los tiempos de espera de conexión, por lo que definir esta propiedad no afectará al controlador. • Las palabras clave de agrupación de conexiones no se tienen en cuenta. Específicamente, esto afecta a las palabras clave siguientes: `Pooling`, `Min Pool Size`, `Max Pool Size`, `Connection Lifetime` y `Connection Reset`.	Todos
OdbcDataReader. GetSchemaTable	ODBC .NET Data Provider no puede recuperar información descriptiva ampliada procedente de servidores que no devuelven esa clase de información. Por tanto, si se conecta a un servidor que no da soporte a información descriptiva ampliada (los servidores afectados), las columnas siguientes de la tabla de metadatos devuelta por `IDataReader.GetSchemaTable()` no son válidas: • `IsReadOnly` • `IsUnique` • `IsAutoIncrement` • `BaseSchemaName` • `BaseCatalogName`	DB2 para OS/390, versión 7 o inferior DB2 para OS/400 DB2 para VM/VSE

Tabla 71. Restricciones de IBM ODBC .NET Data Provider (continuación)

Clase o función	Descripción de la restricción	Servidores DB2 afectados

Procedimientos almacenados	Para llamar a un procedimiento almacenado debe especificar la sintaxis completa de llamada de ODBC.
	
	Por ejemplo, para llamar al procedimiento almacenado `MYPROC` que hace uso de un VARCHAR(10) como parámetro de entrada:
	`[C#]`
	`OdbcCommand cmd = odbcCon.CreateCommand();`
	`cmd.CommandType = CommandType.Text;`
	`cmd.CommandText = "{ CALL MYPROC(?) }"`
	`OdbcParameter p1 = cmd.CreateParameter();`
	`p1.Value = "Joe";`
	`p1.OdbcType = OdbcType.NVarChar;`
	`cmd.Parameters.Add(p1);`
	`cmd.ExecuteNonQuery();`
	
	Nota: Observe que debe utilizar la sintaxis completa de llamada de ODBC aunque esté utilizando `CommandType.StoredProcedure`. Esto está documentado en MSDN, en el tema sobre la propiedad OdbcCommand.CommandText.
		Todos

Procedimientos almacenados: sin nombres de columna para conjuntos de resultados	El servidor DB2 para OS/390 versión 6.1 no devuelve nombres de columna para conjuntos de resultados devueltos desde un procedimiento almacenado. ODBC .NET Data Provider correlaciona estas columnas sin nombre con su número de posición ordinal (por ejemplo, ″1″, ″2″ ″3″). Esto es diferente de la correlación documentada en MSDN: `"Columna1", "Columna2", "Columna3"`.	DB2 para OS/390 versión 6.1
Promoción de índice exclusivo a clave primaria	ODBC .NET Data Provider promociona los índices exclusivos anulables a claves primarias. Esto es contrario a la documentación de MSDN, que establece que los índices exclusivos anulables no se deben promocionar a claves primarias.	Todos

Apéndice A. Visión general de la información técnica de DB2

La información técnica de DB2 está disponible a través de las herramientas y los métodos siguientes:

- Centro de información de DB2
  - Temas (Tareas, concepto y temas de consulta)
  - Ayuda para herramientas de DB2
  - Programas de ejemplo
  - Guías de aprendizaje
- Manuales de DB2
  - Archivos PDF (descargables)
  - Archivos PDF (desde el DVD con PDF de DB2)
  - Manuales en copia impresa
- Ayuda de línea de mandatos
  - Ayuda de mandatos
  - Ayuda de mensajes

Nota: Los temas del Centro de información de DB2 se actualizan con más frecuencia que los manuales en PDF o impresos. Para obtener la información más actualizada, instale las actualizaciones de la documentación cuando estén disponibles, o consulte el Centro de información de DB2 en ibm.com.

Puede acceder a información técnica adicional de DB2 como, por ejemplo, notas técnicas, documentos técnicos y publicaciones IBM Redbooks en línea, en el sitio ibm.com. Acceda al sitio de la biblioteca de software de gestión de información de DB2 en http://www.ibm.com/software/data/sw-library/.

Comentarios sobre la documentación

Agradecemos los comentarios sobre la documentación de DB2. Si tiene sugerencias sobre cómo podemos mejorar la documentación de DB2, envíe un correo electrónico a db2docs@ca.ibm.com. El personal encargado de la documentación de DB2 lee todos los comentarios de los usuarios, pero no puede responderlos directamente. Proporcione ejemplos específicos siempre que sea posible de manera que podamos comprender mejor sus problemas. Si realiza comentarios sobre un tema o archivo de ayuda determinado, incluya el título del tema y el URL.

No utilice esta dirección de correo electrónico para contactar con el Soporte al cliente de DB2. Si tiene un problema técnico de DB2 que no está tratado por la documentación, consulte al centro local de servicio técnico de IBM para obtener ayuda.

Biblioteca técnica de DB2 en copia impresa o en formato PDF

Las tablas siguientes describen la biblioteca de DB2 que está disponible en el Centro de publicaciones de IBM en www.ibm.com/shop/publications/order. Los manuales de DB2 Versión 9.5 en inglés en formato PDF y las versiones traducidas se pueden descargar del sitio www.ibm.com/support/docview.wss?rs=71 &uid=swg2700947.

Aunque las tablas identifican los manuales en copia impresa disponibles, puede que dichos manuales no estén disponibles en su país o región.

El número de documento se incrementa cada vez que se actualiza un manual. Asegúrese de que lee la versión más reciente de los manuales, tal como aparece a continuación:

Nota: El Centro de información de DB2 se actualiza con más frecuencia que los manuales en PDF o impresos.

Tabla 72. Información técnica de DB2

Nombre	Número de documento	Copia impresa disponible
Consulta de las API administrativas	SC11-3505-01	Sí
Rutinas y vistas administrativas	SC11-3507-01	No
Call Level Interface Guide and Reference, Volume 1	SC23-5844-01	Sí
Call Level Interface Guide and Reference, Volume 2	SC23-5845-01	Sí
Consulta de mandatos	SC11-3506-01	Sí
Data Movement Utilities Guide and Reference	SC23-5847-01	Sí
Data Recovery and High Availability Guide and Reference	SC23-5848-01	Sí
Data Servers, Databases, and Database Objects Guide	SC23-5849-01	Sí
Database Security Guide	SC23-5850-01	Sí
Desarrollo de aplicaciones ADO.NET y OLE DB	SC11-3499-01	Sí
Desarrollo de aplicaciones de SQL incorporado	SC11-3500-01	Sí
Desarrollo de aplicaciones Java	SC11-3501-01	Sí
Desarrollo de aplicaciones Perl y PHP	SC11-3502-01	No
Desarrollo de rutinas definidas por el usuario (SQL y externas)	SC11-3503-01	Sí
Iniciación al desarrollo de aplicaciones de bases de datos	GC11-3504-01	Sí
Iniciación a la instalación y administración de DB2 en Linux y Windows	GC11-3511-01	Sí
Internationalization Guide	SC23-5858-01	Sí
Consulta de mensajes, Volumen 1	GI11-7823-00	No
Consulta de mensajes, Volumen 2	GI11-7824-00	No
Guía de migración	GC11-3510-01	Sí
Net Search Extender Guía de administración y del usuario	SC11-3615-01	Sí
Partitioning and Clustering Guide	SC23-5860-01	Sí

Tabla 72. Información técnica de DB2 (continuación)

Nombre	Número de documento	Copia impresa disponible
Query Patroller Administration and User's Guide	SC23-8507-00	Sí
Guía rápida de iniciación para clientes IBM Data Server	GC11-3513-01	No
Guía rápida de iniciación para servidores DB2	GC11-3512-01	Sí
Spatial Extender and Geodetic Data Management Feature Guía del usuario y manual de consulta	SC11-3614-01	Sí
Consulta de SQL, Volumen 1	SC11-3508-01	Sí
Consulta de SQL, Volumen 2	SC11-3509-01	Sí
System Monitor Guide and Reference	SC23-5865-01	Sí
Troubleshooting Guide	GI11-7857-01	No
Tuning Database Performance	SC23-5867-01	Sí
Guía de aprendizaje de Visual Explain	SC11-3518-00	No
Novedades	SC11-3517-01	Sí
Workload Manager Guide and Reference	SC23-5870-01	Sí
pureXML Guide	SC23-5871-01	Sí
XQuery Reference	SC23-5872-01	No

Tabla 73. Información técnica específica de DB2 Connect

Nombre	Número de documento	Copia impresa disponible
Guía rápida de iniciación para DB2 Connect Personal Edition	GC11-3515-01	Sí
Guía rápida de iniciación para servidores DB2 Connect	GC11-3516-01	Sí
Guía del usuario de DB2 Connect	SC11-3514-01	Sí

Tabla 74. Información técnica de Information Integration

Nombre	Número de documento	Copia impresa disponible
Information Integration: Administration Guide for Federated Systems	SC19-1020-01	Sí
Information Integration: ASNCLP Program Reference for Replication and Event Publishing	SC19-1018-02	Sí
Information Integration: Configuration Guide for Federated Data Sources	SC19-1034-01	No
Information Integration: SQL Replication Guide and Reference	SC19-1030-01	Sí

Tabla 74. Información técnica de Information Integration (continuación)

Nombre	Número de documento	Copia impresa disponible
Information Integration: Introduction to Replication and Event Publishing	SC19-1028-01	Sí

Pedido de manuales de DB2 en copia impresa

Si necesita manuales de DB2 en copia impresa, puede comprarlos en línea en varios países o regiones, pero no en todos. Siempre puede hacer pedidos de manuales de DB2 en copia impresa a través del representante local de IBM. Recuerde que algunas publicaciones en copia software del DVD *Documentación en PDF de DB2* no están disponibles en copia impresa. Por ejemplo, no está disponible la publicación *Consulta de mensajes de DB2* en copia impresa.

Las versiones impresas de muchas de las publicaciones de DB2 disponibles en el DVD de Documentación en PDF de DB2 se pueden solicitar a IBM por una cantidad. Dependiendo desde dónde realice el pedido, podrá solicitar manuales en línea, desde el Centro de publicaciones de IBM. Si la realización de pedidos en línea no está disponible en su país o región, siempre puede hacer pedidos de manuales de DB2 en copia impresa al representante local de IBM. Tenga en cuenta que no todas las publicaciones del DVD de Documentación en PDF de DB2 están disponibles en copia impresa.

Nota: La documentación más actualizada y completa de DB2 se conserva en el Centro de información de DB2 en http://publib.boulder.ibm.com/infocenter/db2luw/v9r5.

Para hacer pedidos de manuales de DB2 en copia impresa:

- Para averiguar si puede hacer pedidos de manuales de DB2 en copia impresa en línea en su país o región, consulte el Centro de publicaciones de IBM en el sitio http://www.ibm.com/shop/publications/order. Debe seleccionar un país, región o idioma para poder acceder a la información sobre pedidos de publicaciones y, a continuación, seguir las instrucciones sobre pedidos para su localidad.
- Para hacer pedidos de manuales de DB2 en copia impresa a través del representante local de IBM:
  1. Localice la información de contacto de su representante local desde uno de los siguientes sitios Web:
     - El directorio de IBM de contactos en todo el mundo en el sitio www.ibm.com/planetwide
     - El sitio Web de publicaciones de IBM en el sitio http://www.ibm.com/shop/publications/order. Tendrá que seleccionar su país, región o idioma para acceder a la página de presentación de las publicaciones apropiadas para su localidad. Desde esta página, siga el enlace "Acerca de este sitio".
  2. Cuando llame, indique que desea hacer un pedido de una publicación de DB2.
  3. Proporcione al representante los títulos y números de documento de las publicaciones que desee solicitar. Si desea consultar los títulos y los números de documento, consulte el apartado "Biblioteca técnica de DB2 en copia impresa o en formato PDF" en la página 167.

Visualización de la ayuda para estados de SQL desde el procesador de línea de mandatos

DB2 devuelve un valor de SQLSTATE para las condiciones que pueden ser el resultado de una sentencia de SQL. La ayuda de SQLSTATE explica los significados de los estados de SQL y los códigos de las clases de estados de SQL.

Para invocar la ayuda para estados de SQL, abra el procesador de línea de mandatos y entre:

```
? sqlstate o ? código de clase
```

donde *sqlstate* representa un estado de SQL válido de cinco dígitos y *código de clase* representa los dos primeros dígitos del estado de SQL.

Por ejemplo, `? 08003` visualiza la ayuda para el estado de SQL 08003, y `? 08` visualiza la ayuda para el código de clase 08.

Acceso a diferentes versiones del Centro de información de DB2

Para los temas de DB2 Version 9.5, el URL del Centro de información de DB2 es http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/

Para los temas de DB2 Version 9, el URL del Centro de información de DB2 es http://publib.boulder.ibm.com/infocenter/db2luw/v9/

Para los temas de DB2 Version 8, vaya al URL del Centro de información de la Versión 8 en el sitio: http://publib.boulder.ibm.com/infocenter/db2luw/v8/

Visualización de temas en su idioma preferido en el Centro de información de DB2

El Centro de información de DB2 intenta visualizar los temas en el idioma especificado en las preferencias del navegador. Si un tema no se ha traducido al idioma preferido, el Centro de información de DB2 visualiza dicho tema en inglés.

- Para visualizar temas en su idioma preferido en el navegador Internet Explorer:
  1. En Internet Explorer, pulse en el botón **Herramientas** —> **Opciones de Internet** —> **Idiomas...**. Se abrirá la ventana Preferencias de idioma.
  2. Asegúrese de que su idioma preferido esté especificado como la primera entrada de la lista de idiomas.
     - Para añadir un nuevo idioma a la lista, pulse el botón **Agregar...**.

       Nota: La adición de un idioma no garantiza que el sistema tenga los fonts necesarios para visualizar los temas en el idioma preferido.
     - Para mover un idioma hacia el principio de la lista, seleccione el idioma y pulse el botón **Subir** hasta que el idioma esté en primer lugar en la lista de idiomas.
  3. Borre la antememoria del navegador y, a continuación, renueve la página para visualizar el Centro de información de DB2 en su idioma preferido.
- Para visualizar temas en su idioma preferido en un navegador Firefox o Mozilla:
  1. Seleccione el botón en la sección **Idiomas** del diálogo **Herramientas** —> **Opciones** —> **Avanzado**. Se visualizará el panel Idiomas en la ventana Preferencias.

2. Asegúrese de que su idioma preferido esté especificado como la primera entrada de la lista de idiomas.
   - Para añadir un nuevo idioma a la lista, pulse el botón **Añadir...** a fin de seleccionar un idioma en la ventana Añadir idiomas.
   - Para mover un idioma hacia el principio de la lista, seleccione el idioma y pulse el botón **Subir** hasta que el idioma esté en primer lugar en la lista de idiomas.
3. Borre la antememoria del navegador y, a continuación, renueve la página para visualizar el Centro de información de DB2 en su idioma preferido.

En algunas combinaciones de navegador y sistema operativo, puede que también tenga que cambiar los valores regionales del sistema operativo al entorno local y al idioma de su elección.

Actualización del Centro de información de DB2 instalado en el sistema o en el servidor de intranet

Si ha instalado localmente el Centro de información de DB2, puede obtener las actualizaciones de la documentación de IBM e instalarlas.

Para actualizar el Centro de información de DB2 instalado localmente es preciso que:

1. Detenga el Centro de información de DB2 en el sistema, y reinicie el Centro de información en modalidad autónoma. La ejecución del Centro de información en modalidad autónoma impide que otros usuarios de la red accedan al Centro de información y permite al usuario aplicar las actualizaciones. Los Centros de información no administrativos y no root de DB2 se ejecutan siempre en modalidad autónoma.
2. Utilice la función Actualizar para ver qué actualizaciones están disponibles. Si hay actualizaciones que desee instalar, puede utilizar la función Actualizar para obtenerlas e instalarlas

   Nota: Si su entorno requiere la instalación de actualizaciones del Centro de información de DB2 en una máquina no conectada a Internet, debe duplicar el sitio de actualizaciones en un sistema de archivos local utilizando una máquina que esté conectada a Internet y tenga instalado el Centro de información de DB2. Si muchos usuarios en la red van a instalar las actualizaciones de la documentación, puede reducir el tiempo necesario para realizar las actualizaciones duplicando también el sitio de actualizaciones localmente y creando un proxy para el sitio de actualizaciones.
   Si hay paquetes de actualización disponibles, utilice la característica Actualizar para obtener los paquetes. Sin embargo, la característica Actualizar sólo está disponible en modalidad autónoma.
3. Detenga el Centro de información autónomo y reinicie el Centro de información de DB2 en su equipo.

Nota: En Windows Vista, los mandatos listados más abajo se deben ejecutar como administrador. Para iniciar un indicador de mandatos o una herramienta gráfica con privilegios de administrador completos, pulse con el botón derecho del ratón el atajo y, a continuación, seleccione **Ejecutar como administrador**.

Para actualizar el Centro de información de DB2 instalado en el sistema o en el servidor de Intranet:

1. Detenga el Centro de información de DB2.

- En Windows, pulse **Inicio** → **Panel de control** → **Herramientas administrativas** → **Servicios**. A continuación, pulse con el botón derecho del ratón en el servicio **Centro de información de DB2** y seleccione **Detener**.
- En Linux, especifique el mandato siguiente:

  ```
  /etc/init.d/db2icdv95 stop
  ```

2. Inicie el Centro de información en modalidad autónoma.
   - En Windows:
     a. Abra una ventana de mandatos.
     b. Navegue hasta la vía de acceso en la que está instalado el Centro de información. De forma predeterminada, el Centro de información de DB2 se instala en el directorio <Archivos de programa>\IBM\Centro de información de DB2\Versión 9.5, siendo `<Archivos de programa>` la ubicación del directorio Archivos de programa.
     c. Navegue desde el directorio de instalación al directorio doc\bin.
     d. Ejecute el archivo help_start.bat:

        ```
        help_start.bat
        ```

   - En Linux:
     a. Navegue hasta la vía de acceso en la que está instalado el Centro de información. De forma predeterminada, el Centro de información de DB2 se instala en el directorio /opt/ibm/db2ic/V9.5.
     b. Navegue desde el directorio de instalación al directorio doc/bin.
     c. Ejecute el script help_start:

        ```
        help_start
        ```

   Se inicia el navegador Web por omisión del sistema para visualizar el Centro de información autónomo.
3. Pulse en el botón **Actualizar** (). En la derecha del panel del Centro de información, pulse en **Buscar actualizaciones**. Se visualiza una lista de actualizaciones para la documentación existente.
4. Para iniciar el proceso de instalación, compruebe las selecciones que desee instalar y, a continuación, pulse **Instalar actualizaciones**.
5. Cuando finalice el proceso de instalación, pulse **Finalizar**.
6. Detenga el Centro de información autónomo:
   - En Windows, navegue hasta el directorio doc\bin del directorio de instalación y ejecute el archivo help_end.bat:

     ```
     help_end.bat
     ```

     Nota: El archivo help_end de proceso por lotes contiene los mandatos necesarios para concluir sin peligro los procesos que se iniciaron mediante el archivo help_start de proceso por lotes. No utilice `Control-C` ni ningún otro método para concluir help_start.bat.
   - En Linux, navegue hasta el directorio de instalación doc/bin y ejecute el script help_end:

     ```
     help_end
     ```

     Nota: El script help_end contiene los mandatos necesarios para concluir sin peligro los procesos que se iniciaron mediante el script help_start. No utilice ningún otro método para concluir el script help_start.
7. Reinicie el Centro de información de DB2:

- En Windows, pulse **Inicio → Panel de control → Herramientas administrativas → Servicios**. A continuación, pulse con el botón derecho del ratón en el servicio **Centro de información de DB2** y seleccione **Iniciar**.
- En Linux, especifique el mandato siguiente:

  ```
  /etc/init.d/db2icdv95 start
  ```

El Centro de información de DB2 actualizado visualiza los temas nuevos y actualizados.

Guías de aprendizaje de DB2

Las guías de aprendizaje de DB2 le ayudan a conocer diversos aspectos de productos DB2. Se proporcionan instrucciones paso a paso a través de lecciones.

Antes de comenzar

Puede ver la versión XHTML de la guía de aprendizaje desde el Centro de información en el sitio http://publib.boulder.ibm.com/infocenter/db2help/.

Algunas lecciones utilizan datos o código de ejemplo. Consulte la guía de aprendizaje para obtener una descripción de los prerrequisitos para las tareas específicas.

Guías de aprendizaje de DB2

Para ver la guía de aprendizaje, pulse el título.

"pureXML" en *pureXML Guide*
: Configure una base de datos DB2 para almacenar datos XML y realizar operaciones básicas con el almacén de datos XML nativos.

"Visual Explain" en *Guía de aprendizaje de Visual Explain*
: Analizar, optimizar y ajustar sentencias de SQL para obtener un mejor rendimiento al utilizar Visual Explain.

Información de resolución de problemas de DB2

Existe una gran variedad de información para la resolución y determinación de problemas para ayudarle en la utilización de productos DB2.

Documentación de DB2
: Puede encontrar información sobre la resolución de problemas en la publicación DB2 Troubleshooting Guide o en la sección Soporte y resolución de problemas del Centro de información de DB2. En ellas encontrará información sobre cómo aislar e identificar problemas utilizando herramientas y programas de utilidad de diagnóstico de DB2, soluciones a algunos de los problemas más habituales y otros consejos sobre cómo solucionar problemas que podría encontrar en los productos DB2.

Sitio web de soporte técnico de DB2
: Consulte el sitio Web de soporte técnico de DB2 si tiene problemas y desea obtener ayuda para encontrar las causas y soluciones posibles. El sitio de soporte técnico tiene enlaces a las publicaciones más recientes de DB2, notas técnicas, Informes autorizados de análisis del programa (APAR o arreglos de defectos), fixpacks y otros recursos. Puede buscar en esta base de conocimiento para encontrar posibles soluciones a los problemas.

Acceda al sitio Web de soporte técnico de DB2 en el sitio http://www.ibm.com/software/data/db2/udb/support.html

Términos y condiciones

Los permisos para utilizar estas publicaciones se otorgan sujetos a los siguientes términos y condiciones.

Uso personal: Puede reproducir estas publicaciones para su uso personal, no comercial, siempre y cuando se mantengan los avisos sobre la propiedad. No puede distribuir, visualizar o realizar trabajos derivados de estas publicaciones, o de partes de las mismas, sin el consentimiento expreso de IBM.

Uso comercial: Puede reproducir, distribuir y visualizar estas publicaciones únicamente dentro de su empresa, siempre y cuando se mantengan todos los avisos sobre la propiedad. No puede realizar trabajos derivados de estas publicaciones, ni reproducirlas, distribuirlas o visualizarlas, ni de partes de las mismas fuera de su empresa, sin el consentimiento expreso de IBM.

Excepto lo expresamente concedido en este permiso, no se conceden otros permisos, licencias ni derechos, explícitos o implícitos, sobre las publicaciones ni sobre ninguna información, datos, software u otra propiedad intelectual contenida en el mismo.

IBM se reserva el derecho de retirar los permisos aquí concedidos cuando, a su discreción, el uso de las publicaciones sea en detrimento de su interés o cuando, según determine IBM, las instrucciones anteriores no se cumplan correctamente.

No puede descargar, exportar ni volver a exportar esta información excepto en el caso de cumplimiento total con todas las leyes y regulaciones vigentes, incluyendo todas las leyes y regulaciones sobre exportación de los Estados Unidos.

IBM NO GARANTIZA EL CONTENIDO DE ESTAS PUBLICACIONES. LAS PUBLICACIONES SE PROPORCIONAN ″TAL CUAL″ Y SIN GARANTÍA DE NINGUNA CLASE, NI EXPLÍCITA NI IMPLÍCITA, INCLUYENDO PERO SIN LIMITARSE A LAS GARANTÍAS IMPLÍCITAS DE COMERCIALIZACIÓN, NO VULNERACIÓN E IDONEIDAD PARA UN FIN DETERMINADO.

Apéndice B. Avisos

Esta información ha sido desarrollada para productos y servicios que se ofrecen en Estados Unidos de América

Es posible que IBM no comercialice en otros países algunos productos, servicios o características descritos en este manual. Consulte al representante local de IBM para obtener información sobre los productos y servicios que actualmente pueden adquirirse en su zona. Cualquier referencia a un producto, programa o servicio de IBM no pretende afirmar ni implicar que sólo se pueda utilizar dicho producto, programa o servicio de IBM. En su lugar se puede utilizar cualquier producto, programa o servicio funcionalmente equivalente que no vulnere ninguno de los derechos de propiedad intelectual de IBM. Sin embargo, es responsabilidad del usuario evaluar y verificar el funcionamiento de cualquier producto, programa o servicio que no sea de IBM.

IBM puede tener patentes o solicitudes de patentes en tramitación que afecten al tema tratado en este documento. La posesión de este documento no confiere ninguna licencia sobre dichas patentes. Puede realizar consultas sobre licencias escribiendo a:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
EE.UU.

Para realizar consultas sobre licencias referentes a información de doble byte (DBCS), puede ponerse en contacto con el Departamento de Propiedad Intelectual de IBM de su país/región o escribir a:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokio 106, Japón

El párrafo siguiente no es aplicable al Reino Unido ni a ningún país/región en donde tales disposiciones sean incompatibles con la legislación local:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROPORCIONA ESTA PUBLICACIÓN "TAL CUAL", SIN GARANTÍA DE NINGUNA CLASE, NI EXPLÍCITA NI IMPLÍCITA, INCLUIDAS, PERO SIN LIMITARSE A ELLAS, LAS GARANTÍAS IMPLÍCITAS DE NO VULNERACIÓN DE DERECHOS, COMERCIALIZACIÓN O IDONEIDAD PARA UN FIN DETERMINADO. Algunos estados no permiten la exclusión de garantías expresas o implícitas en determinadas transacciones, por lo que es posible que esta declaración no sea aplicable en su caso.

Esta publicación puede contener inexactitudes técnicas o errores tipográficos. Periódicamente se efectúan cambios en la información aquí contenida; dichos cambios se incorporarán a las nuevas ediciones de la publicación. IBM puede efectuar, en cualquier momento y sin previo aviso, mejoras y cambios en los productos y programas descritos en esta publicación.

Este documento puede proporcionar enlaces o referencias a sitios y recursos que no son de IBM. IBM no representa, no da garantías, ni se compromete con los recursos de terceros ni con los recursos que no son de IBM a los cuales se puede hacer referencia, acceder desde o enlazarse con desde este documento. Un enlace a un sitio que no es de IBM no implica que IBM apruebe el contenido o la utilización de dicho sitio Web o a su propietario. Además, IBM no forma parte ni es responsable de ninguna transacción que el usuario pueda realizar con terceros, aún cuando llegue a conocerlos (o utilice un enlace a ellos) desde un sitio de IBM. De acuerdo a esto, el usuario reconoce y acepta que IBM no es responsable de la disponibilidad de dichos recursos o sitios externos ni tampoco es responsable de ningún contenido, servicio, producto u otros materiales que estén o se encuentren disponibles desde dichos sitios o recursos. Cualquier software que proporcionen terceras partes, estarán sujetos a los términos y condiciones de licencia que acompañen al software.

IBM puede utilizar o distribuir cualquier información que se le facilite de la manera que considere adecuada, sin contraer por ello ninguna obligación con el remitente.

Los licenciatarios de este programa que deseen obtener información sobre él con el fin de habilitar: (i) el intercambio de información entre programas creados de forma independiente y otros programas (incluido éste) y (ii) el uso mutuo de la información intercambiada, deben ponerse en contacto con:

IBM Canada Limited
Office of the Lab Director
8200 Warden Avenue
Markham, Ontario
L6G 1C7
CANADÁ

Dicha información puede estar disponible, sujeta a los términos y condiciones apropiados, incluido en algunos casos el pago de una tarifa.

El programa bajo licencia descrito en este documento y todo el material bajo licencia asociado a él, los proporciona IBM según los términos del Acuerdo de Cliente de IBM, el Acuerdo Internacional de Programas Bajo Licencia de IBM o cualquier acuerdo equivalente entre el usuario e IBM.

Los datos de rendimiento contenidos en este documento se obtuvieron en un entorno controlado. Por lo tanto, los resultados obtenidos en otros entornos operativos pueden variar significativamente. Algunas mediciones pueden haberse realizado en sistemas experimentales y no es seguro que estas mediciones sean las mismas en los sistemas disponibles comercialmente. Además, algunas mediciones pueden haberse calculado mediante extrapolación. Los resultados reales pueden variar. Los usuarios del presente manual deben verificar los datos aplicables para su entorno específico.

La información referente a productos que no son de IBM se ha obtenido de los proveedores de esos productos, de sus anuncios publicados o de otras fuentes disponibles públicamente. IBM no ha probado esos productos y no puede confirmar la exactitud del rendimiento, la compatibilidad ni ninguna otra afirmación referente a productos que no son de IBM. Las preguntas sobre las prestaciones de productos que no son de IBM deben dirigirse a los proveedores de esos productos.

Todas las declaraciones de intenciones de IBM están sujetas a cambio o cancelación sin previo aviso, y sólo representan objetivos.

Este manual puede contener ejemplos de datos e informes que se utilizan en operaciones comerciales diarias. Para ilustrarlos de la forma más completa posible, los ejemplos incluyen nombres de personas, empresas, marcas y productos. Todos estos nombres son ficticios y cualquier similitud con nombres y direcciones utilizados por una empresa real es totalmente fortuita.

LICENCIA DE COPYRIGHT:

Este manual puede contener programas de aplicaciones de ejemplo escritos en lenguaje fuente, que muestran técnicas de programación en diversas plataformas operativas. Puede copiar, modificar y distribuir estos programas de ejemplo como desee, sin pago alguno a IBM con la intención de desarrollar, utilizar, comercializar o distribuir programas de aplicaciones de acuerdo con la interfaz de programación de aplicaciones correspondiente a la plataforma operativa para la que están escritos los programas de ejemplo. Estos ejemplos no se han probado exhaustivamente bajo todas las condiciones. Por lo tanto, IBM no puede asegurar ni implicar la fiabilidad, utilidad o función de estos programas.

Cada copia o parte de estos programas de ejemplo o cualquier trabajo derivado debe incluir una nota de copyright como la siguiente:

© (*nombre de la empresa*) (*año*). Partes de este código proceden de programas de ejemplo de IBM Corp. © Copyright IBM Corp. *_entre el o los años_*. Reservados todos los derechos.

Marcas registradas

Los siguientes términos son marcas registradas de International Business Machines Corporation en los EE.UU. y/o en otros países.

pureXML
MQSeries
DB2
System z9
AISPO
POWER
WebSphere
DB2 Connect
TXSeries
z/OS
System i
AS/400
IBM
zSeries
Rational
Tivoli
OS/400
Approach
pSeries
VisualAge
OpenPower
REXX
AIX
System z
Encina
OS/390
DB2 Universal Database
Redbooks
developerWorks
PowerPC
CICS
SQL/400
Lotus
HACMP
MVS
eServer
ibm.com
iSeries

Los siguientes términos son marcas registradas de otras empresas.

- Linux es una marca registrada de Linus Torvalds en los Estados Unidos y/o en otros países.
- Java y todas las marcas comerciales basadas en Java son marcas comerciales de Sun Microsystems, Inc. en los Estados Unidos y/o en otros países.
- UNIX es una marca registrada de The Open Group en los Estados Unidos y/o en otros países.
- Intel Xeon, Itanium y Pentium e Intel son marcas comerciales de Intel Corporation o de sus subsidiarias en los Estados Unidos y/o en otros países.
- Microsoft y Windows son marcas comerciales de Microsoft Corporation, Inc. en los Estados Unidos y/o en otros países.

Otros nombres de empresas, productos o servicios, pueden ser marcas registradas o marcas de servicio de otras empresas.

Índice

Caracteres Especiales

A

B

C

D

E

F

G

H

I

J

L

M

O

P

R

S

T

V

X

SC11-3499-01

Spine information:

DB2 Versión 9.5 para Linux, UNIX y Windows

Desarrollo de aplicaciones ADO.NET y OLE DB

IBM