Contents

ADO.NET Entity Framework

23/10/2019 • 2 minutes to read • Edit Online

El docs.microsoft.com/ef/ sitio ahora es la ubicación principal para el contenido de Entity Framework.

El contenido de este tema ahora está disponible en la página siguiente: Introducción a Entity Framework.

Información general de Entity Framework

02/07/2020 • 17 minutes to read • Edit Online

La Entity Framework es un conjunto de tecnologías de ADO.NET que admiten el desarrollo de aplicaciones de software orientadas a datos. Los arquitectos y programadores de aplicaciones orientadas a datos se han enfrentado a la necesidad de lograr dos objetivos muy diferentes. Deben modelar las entidades, las relaciones y la lógica de los problemas empresariales que resuelven, y también deben trabajar con los motores de datos que se usan para almacenar y recuperar los datos. Los datos pueden abarcar varios sistemas de almacenamiento, cada uno con sus propios protocolos; incluso las aplicaciones que funcionan con un único sistema de almacenamiento deben equilibrar los requisitos del sistema de almacenamiento con respecto a los requisitos de escribir un código de aplicación eficaz y fácil de mantener.

El Entity Framework permite a los desarrolladores trabajar con datos en forma de objetos y propiedades específicos del dominio, como clientes y direcciones de clientes, sin tener que preocuparse de las tablas y columnas de bases de datos subyacentes en las que se almacenan estos datos. Con Entity Framework, los desarrolladores pueden trabajar en un nivel más alto de abstracción cuando tratan con datos, y pueden crear y mantener aplicaciones orientadas a datos con menos código que en las aplicaciones tradicionales. Dado que el Entity Framework es un componente del .NET Framework, Entity Framework aplicaciones pueden ejecutarse en cualquier equipo en el que se haya instalado el .NET Framework a partir de la versión 3,5 SP1.

Dar vida a los modelos

Un enfoque de diseño habitual para crear una aplicación o un servicio consiste en dividir la aplicación o el servicio en tres partes: un modelo de dominio, un modelo lógico y un modelo físico. El modelo de dominio define las entidades y relaciones del sistema que se está modelando. El modelo lógico de una base de datos relacional normaliza las entidades y relaciones en tablas con restricciones de claves externas. El modelo físico abarca las capacidades de un motor de datos determinado especificando los detalles del almacenamiento en forma de particiones e índices.

Los administradores de bases de datos refinan el modelo físico para mejorar el rendimiento, pero los programadores que escriben el código de la aplicación principalmente se limitan a trabajar con el modelo lógico escribiendo consultas SQL y llamando a procedimientos almacenados. Los modelos de dominio se suelen usar como una herramienta para capturar y comunicar los requisitos de una aplicación, con frecuencia como diagramas inertes que se ven y se explican en las primeras etapas de un proyecto, y a continuación se abandonan. Muchos equipos de desarrolladores omiten la creación de un modelo conceptual y comienzan especificando las tablas, columnas y claves en una base de datos relacional.

El Entity Framework proporciona una vida a los modelos al permitir a los programadores consultar las entidades y relaciones en el modelo de dominio (denominado modelo *conceptual* en el Entity Framework) mientras se confía en el Entity Framework para traducir esas operaciones en comandos específicos del origen de datos. Esto libera a las aplicaciones de las dependencias codificadas de forma rígida en un origen de datos determinado.

Al trabajar con Code First, el modelo conceptual se asigna al modelo de almacenamiento en código. El Entity Framework puede deducir el modelo conceptual en función de los tipos de objeto y las configuraciones adicionales que defina. Los metadatos de asignación se generan durante el tiempo de ejecución basándose en una combinación de cómo se definen los tipos de dominio e información de configuración adicional que se proporciona en código. Entity Framework genera la base de datos según sea necesario en función de los metadatos. Para obtener más información, vea crear un modelo.

Cuando se trabaja con las Herramientas de Entity Data Model, el modelo conceptual, el modelo de almacenamiento y las asignaciones entre los dos se expresan en esquemas basados en XML y se definen en

archivos que tienen extensiones de nombre correspondientes:

- El lenguaje de definición de esquemas conceptuales (CSDL) define el modelo conceptual. CSDL es la implementación del Entity Framework del Entity Data Model. La extensión de archivo es .csdl.
- El lenguaje de definición de esquemas de almacenamiento (SSDL) define el modelo de almacenamiento, que también se denomina modelo lógico. La extensión de archivo es .ssdl.
- El lenguaje de especificación de asignaciones (MSL) define las asignaciones entre los modelos conceptual y de almacenamiento. La extensión de archivo es .msl.

El modelo de almacenamiento y las asignaciones pueden cambiar según sea necesario sin requerir cambios en el modelo conceptual, las clases de datos o el código de la aplicación. Dado que los modelos de almacenamiento son específicos del proveedor, puede trabajar con un modelo conceptual coherente a través de varios orígenes de datos.

El Entity Framework utiliza estos archivos de asignación y modelo para crear, leer, actualizar y eliminar operaciones en las entidades y relaciones del modelo conceptual en las operaciones equivalentes en el origen de datos. La Entity Framework incluso admite la asignación de entidades en el modelo conceptual a procedimientos almacenados en el origen de datos. Para obtener más información, vea las Especificaciones de CSDL, SSDL y MSL.

Asignar objetos a datos

La programación orientada a objetos supone un desafío al interactuar con sistemas de almacenamiento de datos. Aunque la organización de clases suele reflejar la organización de las tablas de bases de datos relacionales, el ajuste no es perfecto. Varias tablas normalizadas suelen corresponder a una sola clase y las relaciones entre las clases se representan a menudo de forma diferente a las relaciones entre tablas. Por ejemplo, para representar el cliente de un pedido de ventas, una clase `Order` podría utilizar una propiedad que contiene una referencia a una instancia de una clase `Customer`, mientras que una fila de la tabla `Order` en una base de datos contiene una columna de clave externa con un valor que corresponde a un valor de clave principal en la tabla `Customer` (o conjunto de columnas). Una clase `Customer` podría tener una propiedad denominada `Orders` que contuviera una colección de instancias de la clase `Order`, mientras que la tabla `Customer` en una base de datos no tiene ninguna columna comparable. El Entity Framework proporciona a los desarrolladores la flexibilidad necesaria para representar relaciones de esta manera, o para modelar más estrechamente las relaciones que se representan en la base de datos.

Las soluciones existentes han intentado cubrir este hueco, que se suele denominar "desigualdad de impedancia", asignando únicamente clases y propiedades orientadas a objetos a las tablas y columnas relacionales. En lugar de tomar este enfoque tradicional, el Entity Framework asigna las tablas relacionales, las columnas y las restricciones Foreign Key de los modelos lógicos a las entidades y relaciones en los modelos conceptuales. Esto permite una mayor flexibilidad al definir los objetos y optimizar el modelo lógico. Las herramientas de Entity Data Model generan clases de datos extensibles basadas en el modelo conceptual. Se trata de clases parciales que se pueden extender con miembros adicionales que el programador agrega. De forma predeterminada, las clases que se generan para un modelo conceptual determinado derivan de las clases base que proporcionan servicios para materializar las entidades como objetos y para realizar un seguimiento de los cambios y guardarlos. Los desarrolladores pueden utilizar estas clases para trabajar con las entidades y relaciones como objetos relacionados mediante asociaciones. Los desarrolladores también pueden personalizar las clases que se generan para un modelo conceptual. Para obtener más información, vea trabajar con objetos.

Obtener acceso y cambiar los datos de la entidad

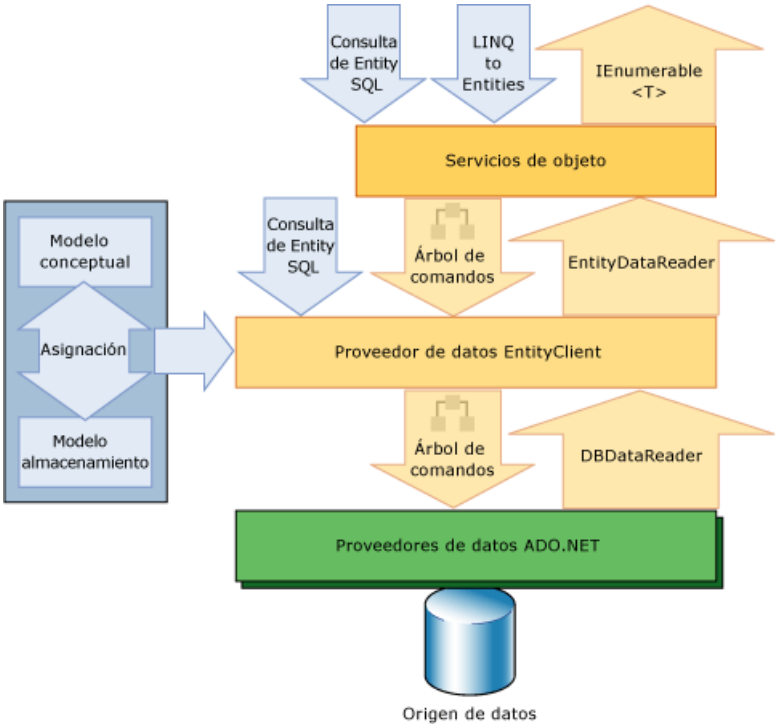
Como algo más que otra solución de asignación objeto-relacional, Entity Framework trata fundamentalmente de permitir que las aplicaciones obtengan acceso y cambien los datos que están representados como entidades y relaciones en el modelo conceptual. En el Entity Framework se usa información en los archivos de modelo y asignación para traducir las consultas de objeto con los tipos de entidad representados en el modelo conceptual

en consultas específicas del origen de datos. Los resultados de la consulta se materializan en objetos que administra el Entity Framework. El Entity Framework proporciona las siguientes formas de consultar un modelo conceptual y devolver objetos:

- LINQ to Entities. Proporciona compatibilidad con Language-Integrated Query (LINQ) para consultar los tipos de entidad que se definen en un modelo conceptual. Para obtener más información, vea LINQ to Entities.
- Entity SQL. Dialecto de SQL independiente del almacenamiento que trabaja directamente con entidades del modelo conceptual y que admite conceptos de Entity Data Model. Entity SQLse utiliza tanto con consultas de objeto como con consultas que se ejecutan con el proveedor de EntityClient. Para obtener más información, vea información general sobre Entity SQL.

El Entity Framework incluye el proveedor de datos de EntityClient. Este proveedor administra las conexiones, traduce las consultas de entidad en consultas específicas del origen de datos y devuelve un lector de datos que el Entity Framework utiliza para materializar los datos de la entidad en los objetos. Cuando no se requiere la materialización de los objetos, el proveedor de EntityClient también se puede utilizar como un proveedor de datos ADO.NET estándar habilitando las aplicaciones para ejecutar las consultas de Entity SQL y usar el lector de datos de solo lectura devuelto. Para obtener más información, consulte Proveedor de EntityClient para Entity Framework.

El diagrama siguiente muestra la arquitectura de Entity Framework para el acceso a datos:



Las herramientas de Entity Data Model pueden generar una clase derivada de `System.Data.Objects.ObjectContext` o `System.Data.Entity.DbContext` que representa el contenedor de entidades en el modelo conceptual. Este contexto del objeto proporciona los medios para realizar el seguimiento de los cambios y administrar las identidades, la simultaneidad y las relaciones. Esta clase también expone un método `SaveChanges` que escribe las inserciones, actualizaciones y eliminaciones en el origen de datos. Al igual que las consultas, estas modificaciones son realizadas bien por los comandos que el sistema genera automáticamente o bien por los procedimientos almacenados que el programador especifica.

Proveedores de datos

El `EntityClient` proveedor extiende el modelo de proveedor ADO.net accediendo a los datos en términos de entidades y relaciones conceptuales. Ejecuta consultas que utilizan Entity SQL. Entity SQL proporciona el lenguaje

de consultas subyacente que permite a `EntityClient` comunicarse con la base de datos. Para obtener más información, consulte Proveedor de EntityClient para Entity Framework.

El Entity Framework incluye un proveedor de datos SqlClient actualizado que admite árboles de comandos canónicos. Para obtener más información, vea SqlClient para el Entity Framework.

Herramientas de Entity Data Model

Junto con el tiempo de ejecución de Entity Framework, Visual Studio incluye las herramientas de asignación y modelado. Para obtener más información, vea modelado y asignación.

Saber más

Para obtener más información sobre el Entity Framework, consulte:

Introducción : proporciona información sobre cómo ponerse en marcha rápidamente con la guía de Inicio rápido, que muestra cómo crear una sencilla aplicación de Entity Framework.

Entity Framework terminología : define muchos de los términos introducidos por el Entity Data Model y el Entity Framework y que se usan en la documentación de Entity Framework.

Recursos de Entity Framework : proporciona vínculos a temas conceptuales y vínculos a temas externos y recursos para crear aplicaciones de Entity Framework.

Consulte también

- ADO.NET Entity Framework

Modelado y asignación

20/02/2020 • 2 minutes to read • Edit Online

En el Entity Framework, puede definir el modelo conceptual, el modelo de almacenamiento y la asignación entre los dos de la forma que mejor se adapte a su aplicación. Las herramientas de Entity Data Model de Visual Studio le permiten crear un. archivo edmx de una base de datos o un modelo gráfico y, a continuación, actualice el archivo cuando cambie la base de datos o el modelo.

A partir de Entity Framework 4.0.1 también puede crear un modelo mediante programación usando desarrollo Code First. Hay dos escenarios diferentes para el desarrollo Code First. En ambos casos, el desarrollador define un modelo codificando definiciones de clase de .NET Framework y especifica opcionalmente la asignación o configuración adicional usando anotaciones de datos o la API fluida.

Para obtener más información, vea crear un modelo.

También puede usar el generador de EDM, que se incluye con el .NET Framework. EdmGen.exe genera los archivos .csdl, .ssdl y .msl a partir de un origen de datos existente. También puede crear manualmente el contenido del modelo y la asignación. Para obtener más información, vea generador de EDM (EdmGen. exe).

Generador de EDM (EdmGen.exe)

08/11/2019 • 11 minutes to read • Edit Online

EdmGen. exe es una herramienta de línea de comandos que se usa para trabajar con Entity Framework los archivos de modelo y asignación. Puede utilizar la herramienta EdmGen.exe para lo siguiente:

- Conectarse a un origen de datos mediante un proveedor de datos de .NET Framework específico del origen de datos y generar los archivos de modelo conceptual (. CSDL), de modelo de almacenamiento (. SSDL) y de asignación (. MSL) utilizados por el Entity Framework. Para obtener más información, vea Cómo: usar EdmGen. exe para generar los archivos de asignación y de modelo.
- Validar un modelo existente. Para obtener más información, consulte Cómo: usar EdmGen. exe para validar los archivos de asignación y de modelo.
- Generar un o archivo de código de C# o Visual Basic que contenga las clases de objetos generados a partir de un archivo de modelo conceptual (.csdl). Para obtener más información, vea Cómo: usar EdmGen. exe para generar código de nivel de objeto.
- Generar un archivo de código de C# o Visual Basic que contenga las vistas generadas previamente para un modelo existente. Para obtener más información, Cómo: generar previamente vistas para mejorar el rendimiento de las consultas.

La herramienta EdmGen. exe se instala en el directorio .NET Framework. En muchos casos, se encuentra en C:\windows\Microsoft.NET\Framework\v4.0. Para los sistemas de 64 bits, se encuentra en C:\windows\Microsoft.NET\Framework64\v4.0. También puede tener acceso a la herramienta EdmGen. exe desde el símbolo del sistema de Visual Studio (haga clic en **Inicio**, seleccione **todos los programas**, **Microsoft Visual Studio 2010**, seleccione **Visual Studio Tools**y, a continuación, haga clic en **Visual Studio 2010 Símbolo del sistema**).

Sintaxis

```
EdmGen /mode:choice [options]
```

Modo

Cuando use la herramienta EdmGen.exe, deberá especificar uno de los modos siguientes.

MODO	DESCRIPCIÓN
`/mode:ValidateArtifacts`	Valida los archivos .csdl, .ssdl y .msl, y muestra los errores o advertencias. Esta opción requiere al menos uno de los argumentos `/inssdl` o `/incsdl`. Si se especifica `/inmsl`, también se requieren los argumentos `/inssdl` y `/incsdl`.

MODO	DESCRIPCIÓN
`/mode:FullGeneration`	Utiliza la información de la conexión a bases de datos especificada en la opción `/connectionstring` y genera los archivos .csdl, .ssdl, .msl, de nivel de objetos y de vista. Esta opción requiere un argumento `/connectionstring` y un argumento `/project` o bien argumentos `/outssdl`, `/outcsdl`, `/outmsdl`, `/outobjectlayer`, `/outviews`, `/namespace` y `/entitycontainer`.
`/mode:FromSSDLGeneration`	Genera los archivos .csdl y .msl, el código fuente y las vistas a partir del archivo .ssdl especificado. Esta opción requiere el argumento `/inssdl` y un argumento `/project`, o los argumentos `/outcsdl`, `/outmsl`, `/outobjectlayer`, `/outviews`, `/namespace,` y `/entitycontainer`.
`/mode:EntityClassGeneration`	Crea un archivo de código fuente que contiene las clases que se generaron a partir del archivo .cdsl. Esta opción requiere el argumento `/incsdl` y el argumento `/project` u `/outobjectlayer`. El argumento `/language` es opcional.
`/mode:ViewGeneration`	Crea un archivo de código fuente que contiene las vistas generadas a partir de los archivos .csdl, .ssdl y .msl. Esta opción requiere los argumentos `/inssdl`, `/incsdl`, `/inmsl,` y `/project` u `/outviews`. El argumento `/language` es opcional.

Opciones

OPCIÓN	DESCRIPCIÓN
`/p[roject]:` cadena de <	Especifica el nombre del proyecto que se usará. El nombre del proyecto se utiliza como valor predeterminado para la configuración del espacio de nombres, el nombre de los archivos de modelo y asignación, el nombre del archivo de origen del objeto y el nombre del archivo de código fuente de la generación de las vistas. El nombre del contenedor de entidades se establece en <contexto de > del proyecto.
`/prov[ider]:` cadena de <	Nombre del proveedor de datos .NET Framework que se va a utilizar para generar el archivo del modelo de almacenamiento (.ssdl). El proveedor predeterminado es el Proveedor de datos .NET Framework para SQL Server (System.Data.SqlClient).
`/c[onnectionstring]:` cadena de conexión de <	Especifica la cadena que se utiliza para conectarse al origen de datos.

OPCIÓN	DESCRIPCIÓN
`/incsdl:` <archivo >	Especifica el archivo .csdl o un directorio donde se encuentran los archivos .csdl. Se puede especificar este argumento varias veces para poder especificar varios directorios o archivos .csdl. Especificar varios directorios puede ser útil para generar las clases (`/mode:EntityClassGeneration`) o las vistas (`/mode:ViewGeneration`) cuando el modelo conceptual se divide en varios archivos. Esto también puede ser útil si se desea validar varios modelos (`/mode:ValidateArtifacts`).
`/refcsdl:` <archivo >	Especifica el archivo .csdl adicional o los archivos que se usan para resolver las referencias en el archivo .csdl de origen. (El archivo .csdl de origen es el archivo que determina la opción `/incsdl`). El archivo `/refcsdl` contiene los tipos de los que el archivo .csdl de origen depende. Este argumento se puede especificar varias veces.
`/inmsl:` <archivo >	Especifica el archivo .msl o un directorio donde se encuentran los archivos .msl. Este argumento se puede especificar varias veces para poder especificar varios directorios o archivos .msl. Especificar varios directorios puede ser útil para generar las vistas (`/mode:ViewGeneration`) cuando el modelo conceptual se divide en varios archivos. Esto también puede ser útil si se desea validar varios modelos (`/mode:ValidateArtifacts`).
`/inssdl:` <archivo >	Especifica el archivo .ssdl o un directorio donde se encuentra. Se puede especificar este argumento varias veces para poder especificar varios directorios o archivos .ssdl. Esto puede ser útil si se desea validar varios modelos `(/mode:ValidateArtifacts)` .
`/outcsdl:` <archivo >	Especifica el nombre del archivo .csdl que se creará.
`/outmsl:` <archivo >	Especifica el nombre del archivo .msl que se creará.
`/outssdl:` <archivo >	Especifica el nombre del archivo .ssdl que se creará.
`/outobjectlayer:` <archivo >	Especifica el nombre del archivo de código fuente que contiene los objetos generados a partir del archivo .csdl.
`/outviews:` <archivo >	Especifica el nombre del archivo de código fuente que contiene las vistas que se generaron.
`/language:` [VB|CSharp]	Especifica el lenguaje de los archivos de código fuente generados. El lenguaje predeterminado es C#.
`/namespace:` cadena de<	Especifica el espacio de nombres del modelo que se va a utilizar. El espacio de nombres se establece en el archivo .csdl al ejecutar `/mode:FullGeneration` o `/mode:FromSSDLGeneration` . El espacio de nombres no se usa al ejecutar `/mode:EntityClassGeneration` .
`/entitycontainer:` cadena de<	Especifica el nombre que se va a aplicar al elemento `<EntityContainer>` en los archivos de modelo y asignación generados.

OPCIÓN	DESCRIPCIÓN
`/pl[uralize]`	Aplica las reglas del idioma inglés para singulares y plurales a los nombres de `Entity`, `EntitySet` y `NavigationProperty` en el modelo conceptual. Esta opción realizará las siguientes acciones: : Convierte todos los nombres de `EntityType` en singular. : Poner todos los nombres de `EntitySet` en plural. -Para cada `NavigationProperty` que devuelve como máximo una entidad, cree el nombre singular. -Para cada `NavigationProperty` que devuelve más de una entidad, convierta el nombre en plural.
`/SuppressForeignKeyProperties or /nofk`	Evita que las columnas de clave externa se expongan como propiedades escalares en tipos de entidad en el modelo conceptual.
`/help` o `?`	Muestra las opciones y la sintaxis de los comandos para la herramienta.
`/nologo`	Evita que se muestre el mensaje de copyright.
`/targetversion:` cadena de <	La versión de .NET Framework que se usará para compilar el código generado. Las versiones admitidas son 4 y 4.5. El valor predeterminado es 4.

En esta sección

Uso de EdmGen.exe para generar los archivos de asignación y de modelo

Uso de EdmGen.exe para generar código de capa de objeto

Uso de EdmGen.exe para validar los archivos de asignación y de modelo

Vea también

- ADO.NET Entity Data Model herramientas
- Entity Data Model
- Especificaciones CSDL, SSDL MSL

Cómo: Usar EdmGen.exe para generar los archivos de asignación y de modelo

21/03/2020 • 2 minutes to read • Edit Online

En este tema se muestra cómo usar la herramienta EDM Generator (EdmGen.exe) para generar los siguientes archivos basados en la base de datos School:

- Un modelo conceptual (un archivo .csdl).
- Un modelo de almacenamiento (un archivo .ssdl).
- Asignación entre los modelos conceptual y de almacenamiento (un archivo .msl).
- Código del nivel de objeto en Visual Basic o C#.
- Archivos de vistas.

La herramienta EdmGen.exe utiliza /mode:FullGeneration para generar los archivos enumerados anteriormente. Para obtener más información acerca de los comandos EdmGen.exe, vea Generador de EDM (EdmGen.exe).

Si usa EdmGen.exe para generar el modelo y los archivos de asignación, debe configurar el proyecto de Visual Studio para usar Entity Framework. Para obtener más información, vea Cómo: configurar manualmente un proyectode Entity Framework .

> **NOTE**
> Un modelo conceptual generado mediante EdmGen.exe incluye todos los objetos de la base de datos. Si desea generar un modelo conceptual que solo incluya objetos específicos, use el asistente de Entity Data Model. Para obtener más información, vea Cómo: usar el Asistente para Entity Data Model.

Para generar el modelo School para un proyecto de Visual Basic con EdmGen.exe

1. Cree la base de datos School. Para obtener más información, consulte Creación de la base dedatos de ejemplo escolar .
2. En el símbolo del sistema, ejecute el comando siguiente sin los saltos de línea:

```
"%windir%\Microsoft.NET\Framework\v4.0.30319\edmgen.exe" /mode:fullgeneration
/c:"Data Source=%datasourceserver%; Initial Catalog=School; Integrated Security=SSPI"
/project:School /entitycontainer:SchoolEntities /namespace:SchoolModel /language:VB
```

Para generar el modelo School para un proyecto de C# con EdmGen.exe

1. Cree la base de datos School. Para obtener más información, consulte Creación de la base dedatos de ejemplo escolar .
2. En el símbolo del sistema, ejecute el comando siguiente sin los saltos de línea:

```
"%windir%\Microsoft.NET\Framework\v4.0.30319\edmgen.exe" /mode:fullgeneration
/c:"Data Source=%datasourceserver%; Initial Catalog=School; Integrated Security=SSPI"
/project:School /entitycontainer:SchoolEntities /namespace:SchoolModel /language:CSharp
```

Consulte también

- Modelado y asignación
- Cómo: Configurar manualmente un proyecto de Entity Framework
- Cómo: Generar previamente vistas para mejorar el rendimiento de la consulta
- Herramientas de ADO.NET Entity Data Model
- Cómo: Usar EdmGen.exe para validar los archivos de asignación y de modelo

Cómo: Usar EdmGen.exe para generar código de capa de objeto

21/03/2020 • 2 minutes to read • Edit Online

En este tema se muestra cómo utilizar la herramienta Generador de EDM (EdmGen.exe) para generar código de capa de objeto basado en el archivo .csdl.

Para generar código del nivel de objeto para el modelo School en un proyecto de Visual Basic utilizando EdmGen.exe

1. Cree la base de datos School. Para obtener más información, consulte Creación de la base dedatos de ejemplo escolar .
2. Genere el modelo School u obtenga el archivo School.csdl. Para obtener más información, vea Cómo: Usar EdmGen.exe para generar el modelo ylos archivos de asignación .
3. En el símbolo del sistema, ejecute el comando siguiente sin los saltos de línea:

```
"%windir%\Microsoft.NET\Framework\v4.0.30319\edmgen.exe" /mode:EntityClassGeneration
/incsdl:.\School.csdl /outobjectlayer:.\School.Objects.vb /language:VB
```

Para generar código del nivel de objeto para el modelo School en un proyecto de C# utilizando EdmGen.exe

1. Cree la base de datos School. Para obtener más información, consulte Creación de la base dedatos de ejemplo escolar .
2. Genere el modelo School u obtenga el archivo School.csdl. Para obtener más información, vea Cómo: Usar EdmGen.exe para generar el modelo ylos archivos de asignación .
3. En el símbolo del sistema, ejecute el comando siguiente sin los saltos de línea:

```
"%windir%\Microsoft.NET\Framework\v4.0.30319\edmgen.exe" /mode:EntityClassGeneration
/incsdl:.\School.csdl /outobjectlayer:.\School.Objects.cs /language:CSharp
```

Consulte también

- Modelado y asignación
- Cómo: Configurar manualmente un proyecto de Entity Framework
- Herramientas de Entity Data Model de ADO.NET
- Cómo: Generar previamente vistas para mejorar el rendimiento de la consulta
- Cómo: Usar EdmGen.exe para generar los archivos de asignación y de modelo

Procedimiento para usar EdmGen.exe para validar los archivos de asignación y de modelo

23/10/2019 • 2 minutes to read • Edit Online

En este tema se muestra cómo usar la herramienta generador de EDM (EdmGen. exe) para validar los archivos de asignación y de modelo. Para obtener más información, vea Entity Data Model.

Para validar el modelo School mediante el uso de EdmGen.exe

1. Cree la base de datos School. Para obtener más información, vea crear la base de datos de ejemplo School.
2. Genere el modelo School. Para obtener más información, consulte Cómo Use EdmGen. exe para generar los archivosde asignación y de modelo.
3. En el símbolo del sistema, ejecute el comando siguiente sin los saltos de línea:

```
"%windir%\Microsoft.NET\Framework\v4.0.30319\edmgen.exe" /mode:ValidateArtifacts /inssdl:.\School.ssdl
/inmsl:.\School.msl /incsdl:.\School.csdl
```

Vea también

- Cómo: Configurar manualmente un proyecto de Entity Framework
- ADO.NET Entity Data Model herramientas
- Procedimientos: Generar previamente vistas para mejorar el rendimiento de las consultas
- Cómo: Usar EdmGen. exe para generar código de nivel de objeto

Cómo: Definir la cadena de conexión

21/03/2020 • 2 minutes to read • Edit Online

En este tema se muestra cómo definir la cadena de conexión que se usa al conectarse a un modelo conceptual. Este tema se basa en el modelo conceptual AdventureWorks Sales. El modelo de ventas AdventureWorks se utiliza en todos los temas relacionados con tareas en la documentación de Entity Framework. En este tema se supone que ya ha configurado Entity Framework y definido el modelo de ventas de AdventureWorks. Para obtener más información, consulte Cómo: Definir manualmente el modelo ylos archivos de asignación . Los procedimientos de este tema también se incluyen en Cómo: configurar manualmente un proyectode Entity Framework .

> **NOTE**
> Si usa el Asistente para Entity Data Model en un proyecto de Visual Studio, genera automáticamente un archivo .edmx y configura el proyecto para usar Entity Framework. Para obtener más información, vea Cómo: usar el Asistente para Entity Data Model.

Para definir la cadena de conexión de Entity Framework

- Abra el archivo de configuración de la aplicación del proyecto (app.config) y, a continuación, agregue la siguiente cadena de conexión:

```
<connectionStrings>
    <add name="AdventureWorksEntities"
         connectionString="metadata=.\AdventureWorks.csdl|.\AdventureWorks.ssdl|.\AdventureWorks.msl;
         provider=System.Data.SqlClient;provider connection string='Data Source=localhost;
         Initial Catalog=AdventureWorks;Integrated Security=True;Connection Timeout=60;
         multipleactiveresultsets=true'" providerName="System.Data.EntityClient" />
</connectionStrings>
```

Si el proyecto no tiene un archivo de configuración de aplicación, puede agregar uno seleccionando **Agregar nuevo elemento** en el menú **Proyecto,** seleccionando la categoría **General,** seleccionando Archivo de **configuración**de la aplicación y, a continuación, haciendo clic en **Agregar**.

Consulte también

- Quickstart
- Cómo: Crear un nuevo archivo .edmx
- Herramientas de Entity Data Model de ADO.NET

Procedimiento para hacer que los archivos de asignación y de modelo sean recursos incrustados

23/10/2019 • 2 minutes to read • Edit Online

El Entity Framework permite implementar archivos de asignación y de modelo como recursos incrustados de una aplicación. El ensamblado con los archivos de asignación y de modelo incrustados se debe cargar en el mismo dominio de aplicación que la conexión de entidad. Para más información, consulte Cadenas de conexión. De forma predeterminada, las herramientas de Entity Data Model insertan los archivos de asignación y de modelo. Al definir manualmente los archivos de asignación y de modelo, use este procedimiento para asegurarse de que los archivos se implementan como recursos incrustados junto con una aplicación Entity Framework.

> **NOTE**
> Para mantener los recursos incrustados, deberá repetir este procedimiento cada vez que se modifiquen los archivos de asignación y de modelo.

Para incrustar los archivos de asignación y de modelo

1. En **Explorador de soluciones**, seleccione el archivo conceptual (. CSDL).
2. En el panel **propiedades** , establezca **acción de compilación** en **recurso incrustado**.
3. Repita los pasos 1 y 2 para el archivo de almacenamiento (.ssdl) y el archivo de asignación (.msl).
4. En **Explorador de soluciones**, haga doble clic en el archivo app. config y, a `Metadata` continuación, modifique `connectionString` el parámetro en el atributo basándose en uno de los siguientes formatos:
   - `Metadata=``res://<assemblyFullName>/<resourceName>;`
   - `Metadata=``res://*/<resourceName>;`
   - `Metadata=res://*;`

   Para más información, consulte Cadenas de conexión.

Ejemplo

La siguiente cadena de conexión hace referencia a los archivos de asignación y de modelo incrustados para el modelo AdventureWorks Sales. Esta cadena de conexión está almacenada en el archivo App.config del proyecto.

Vea también

- Modelado y asignación
- Cómo: Definir la cadena de conexión
- Procedimientos: Compilación de una cadena de conexión de EntityConnection
- ADO.NET Entity Data Model herramientas

Trabajar con lenguaje de definición de datos

23/10/2019 • 4 minutes to read • Edit Online

A partir de la versión 4 de .NET Framework, el Entity Framework admite el lenguaje de definición de datos (DDL). Esto le permite crear o eliminar una instancia de la base de datos basada en la cadena de conexión y los metadatos del modelo de almacenamiento (SSDL).

Los siguientes métodos de ObjectContext utilizan la cadena de conexión y el contenido SSDL para crear o eliminar la base de datos, comprobar si la base de datos existe y ver el script DDL generado:

- CreateDatabase
- DeleteDatabase
- DatabaseExists
- CreateDatabaseScript

> **NOTE**
> La ejecución de los comandos DDL supone que se cuenta con los permisos necesarios.

Los métodos enumerados anteriormente delegan la mayoría del trabajo en el proveedor de datos de ADO.NET subyacente. Es responsabilidad del proveedor asegurarse de que la convención de nomenclatura utilizada para generar los objetos de base de datos es coherente con las convenciones utilizadas para las consultas y las actualizaciones.

En el siguiente ejemplo se muestra cómo generar la base de datos en función del modelo existente. También agrega un nuevo objeto entidad al contexto del objeto y, a continuación, lo guarda en la base de datos.

Procedimientos

Para definir una base de datos en función del modelo existente

1. Cree una aplicación de consola.
2. Agregue un modelo existente a la aplicación.
   a. Agregue un modelo vacío denominado `SchoolModel`. Para crear un modelo vacío, vea el procedimiento: Cree un nuevo tema del archivo . edmx.

   El archivo SchoolModel.edmx se añade al proyecto.

   a. Copie el contenido conceptual, de almacenamiento y de asignación para el modelo School en el tema School Model .

   b. Abra el archivo SchoolModel.edmx y pegue el contenido dentro de las etiquetas `edmx:Runtime`.
3. Agregue el siguiente código a la función principal. El código inicializa la cadena de conexión en el servidor de bases de datos, ve el script DDL, crea la base de datos, agrega una nueva entidad al contexto y guarda los cambios en la base de datos.

```
// Initialize the connection string.
String connectionString =
"metadata=res://*/School.csdl|res://*/School.ssdl|res://*/School.msl;provider=System.Data.SqlClient;" +
"provider connection string=\"Data Source=.;Initial Catalog=School;Integrated
Security=True;MultipleActiveResultSets=True\"";

using (SchoolEntities context = new SchoolEntities(connectionString))
{
    try
    {
        if (context.DatabaseExists())
        {
            // Make sure the database instance is closed.
            context.DeleteDatabase();
        }
        // View the database creation script.
        Console.WriteLine(context.CreateDatabaseScript());
        // Create the new database instance based on the storage (SSDL) section
        // of the .edmx file.
        context.CreateDatabase();

        // The following code adds a new objects to the context
        // and saves the changes to the database.
        Department dpt = new Department
        {
            Name = "Engineering",
            Budget = 350000.00M,
            StartDate = DateTime.Now
        };

        context.Departments.AddObject(dpt);
        // An entity has a temporary key
        // until it is saved to the database.
        Console.WriteLine(dpt.EntityKey.IsTemporary);
        context.SaveChanges();
        // The object was saved and the key
        // is not temporary any more.
        Console.WriteLine(dpt.EntityKey.IsTemporary);
    }
    catch (InvalidOperationException ex)
    {
        Console.WriteLine(ex.InnerException.Message);
    }
    catch (NotSupportedException ex)
    {
        Console.WriteLine(ex.InnerException.Message);
    }
 }
```

```
' Initialize the connection string.
Dim connectionString As String = _

"metadata=res://*/School.csdl|res://*/School.ssdl|res://*/School.msl;provider=System.Data.SqlClient;" &
_
    "provider connection string=""Data Source=.;Initial Catalog=School;Integrated
Security=True;MultipleActiveResultSets=True"""

Using context As New SchoolEntities(connectionString)
    Try
        If context.DatabaseExists() Then
            ' Make sure the database instance is closed.
            context.DeleteDatabase()
        End If
        ' View the database creation script.
        Console.WriteLine(context.CreateDatabaseScript())
        ' Create the new database instance based on the storage (SSDL) section
        ' of the .edmx file.
        context.CreateDatabase()

        ' The following code adds a new objects to the context
        ' and saves the changes to the database.
        Dim dpt As New Department()

        context.Departments.AddObject(dpt)
        ' An entity has a temporary key
        ' until it is saved to the database.
        Console.WriteLine(dpt.EntityKey.IsTemporary)
        context.SaveChanges()

        ' The object was saved and the key
        ' is not temporary any more.
        Console.WriteLine(dpt.EntityKey.IsTemporary)

    Catch ex As InvalidOperationException
        Console.WriteLine(ex.InnerException.Message)
    Catch ex As NotSupportedException
        Console.WriteLine(ex.InnerException.Message)
    End Try
End Using
```

Consultar un modelo conceptual

20/02/2020 • 2 minutes to read • Edit Online

El Entity Framework ADO.NET permite consultar un modelo conceptual. Para consultar el modelo conceptual con la versión más reciente del Entity Framework, vea consultar datos.

Trabajar con objetos

20/02/2020 • 2 minutes to read • Edit Online

El Entity Framework permite consultar, insertar, actualizar y eliminar datos, que se expresan como objetos con tipo Common Language Runtime (CLR) que son instancias de tipos de entidad. Los tipos de entidad representan las entidades definidas en el modelo conceptual. El Entity Framework asigna las entidades y relaciones que se definen en un modelo conceptual a un origen de datos. El Entity Framework proporciona funciones para realizar lo siguiente: materializar los datos devueltos del origen de datos como objetos; realizar un seguimiento de los cambios realizados en los objetos; controlar la simultaneidad; propagar los cambios de los objetos de nuevo al origen de datos; y enlazar objetos a controles.

Para obtener más información acerca de cómo trabajar con objetos en la última versión del Entity Framework vea trabajar con objetos.

Trabajar con proveedores de datos

23/10/2019 • 2 minutes to read • Edit Online

Los temas de esta sección describen los servicios y los proveedores que transforman las consultas sobre un modelo conceptual en consultas nativas sobre un origen de datos que Entity Framework admite.

En esta sección

Cadenas de conexión

Proveedores de datos de Entity Framework

Escritura de un proveedor de datos de Entity Framework

Vea también

- Lenguaje Entity SQL
- ADO.NET Entity Framework

Cadenas de conexión en el Entity Framework ADO.NET

02/07/2020 • 14 minutes to read • Edit Online

Una cadena de conexión contiene información de inicialización que se transfiere como un parámetro desde un proveedor de datos a un origen de datos. La sintaxis depende del proveedor de datos y la cadena de conexión se analiza mientras se intenta abrir una conexión. Las cadenas de conexión que usa Entity Framework contienen la información que se emplea para conectar con el proveedor de datos ADO.NET subyacente que Entity Framework admite. También contienen información sobre los archivos del modelo y de asignación necesarios.

El proveedor de EntityClient utiliza la cadena de conexión al obtener acceso a los metadatos del modelo y de asignación y al conectar con el origen de datos. Se puede obtener acceso a la cadena de conexión o establecerse a través de la propiedad ConnectionString de EntityConnection. La clase EntityConnectionStringBuilder se puede utilizar para construir mediante programación los parámetros de la cadena de conexión o tener acceso a ellos. Para obtener más información, vea Cómo: compilar una cadena de conexión de EntityConnection.

Las herramientas de Entity Data Model generan una cadena de conexión que se almacena en el archivo de configuración de la aplicación. ObjectContext recupera esta información de conexión automáticamente al crear consultas de objetos. Se puede tener acceso al elemento EntityConnection que usa una instancia de ObjectContext desde la propiedad Connection. Para obtener más información, consulte Administración de conexiones y transacciones.

Sintaxis de cadenas de conexión

Para obtener información sobre la sintaxis general de las cadenas de conexión, vea Sintaxis de cadena de conexión | Cadenas de conexión en ADO.NET.

Parámetros de la cadena de conexión

En la tabla siguiente se muestran los nombres válidos para los valores de palabra clave en la propiedad ConnectionString.

PALABRA CLAVE	DESCRIPCIÓN
`Provider`	Se requiere si no se especifica la palabra clave `Name`. El nombre del proveedor, que se usa para recuperar el objeto DbProviderFactory para el proveedor subyacente. Este valor es constante. Cuando la palabra clave `Name` no se incluye en una cadena de conexión de Entity, se requiere un valor no vacío para la palabra clave `Provider`. Esta palabra clave y la palabra clave `Name` se excluyen mutuamente.

PALABRA CLAVE	DESCRIPCIÓN
`Provider Connection String`	Opcional. Especifica la cadena de conexión específica del proveedor que se pasa al origen de datos subyacente. Esta cadena de conexión contiene pares de palabra clave y valor válidos para el proveedor de datos. Un valor de `Provider Connection String` no válido provocará un error en tiempo de ejecución cuando sea evaluado por el origen de datos. Esta palabra clave y la palabra clave `Name` se excluyen mutuamente. Asegúrese de omitir el valor de acuerdo con la sintaxis general de las cadenas de conexión de ADO.net. Considere, por ejemplo, la siguiente cadena de conexión: `Server=serverName; User ID = userID` . Debe ser un carácter de escape porque contiene un punto y coma. Como no contiene comillas dobles, se pueden usar para el escape: `Provider Connection String ="Server=serverName; User ID = userID";`

`Metadata`	Se requiere si no se especifica la palabra clave `Name`. Una lista delimitada por barras verticales de los directorios, archivos y localizadores de recursos en que se ha de buscar información de asignación y metadatos. A continuación se muestra un ejemplo:
	
	`Metadata=`
	
	`c:\model &#124; c:\model\sql\mapping.msl;`
	
	Los espacios en blanco a cada lado del separador de barra vertical se pasan por alto.
	
	Esta palabra clave y la palabra clave `Name` se excluyen mutuamente.

`Name`	La aplicación puede especificar, si se desea, el nombre de conexión en un archivo de configuración de la aplicación que proporcione los valores de cadena de conexión con pares palabra clave-valor necesarios. En este caso, no es posible suministrarlos directamente en la cadena de conexión. La palabra clave `Name` no se permite en un archivo de configuración. Cuando la palabra clave `Name` no se incluye en la cadena de conexión, se requiere un valor no vacío para la palabra clave Provider. Esta palabra clave y todas las demás palabras clave de cadena de conexión se excluyen mutuamente.

El siguiente es un ejemplo de una cadena de conexión para el modelo AdventureWorks Sales almacenado en el archivo de configuración de la aplicación:

Ubicaciones de los archivos del modelo y de asignación

El parámetro `Metadata` contiene una lista de ubicaciones en las que el proveedor `EntityClient` busca los archivos del modelo y de asignación. Los archivos del modelo y de asignación se suelen implementar en el

mismo directorio que el archivo ejecutable de la aplicación. Estos archivos también se pueden implementar en una ubicación concreta o incluirse como un recurso incrustado en la aplicación.

Los recursos incrustados se especifican como sigue:

```
Metadata=res://<assemblyFullName>/<resourceName>
```

Las opciones siguientes están disponibles para definir la ubicación de un recurso incrustado:

OPCIÓN	DESCRIPCIÓN
`assemblyFullName`	Nombre completo de un ensamblado con el recurso incrustado. El nombre incluye el nombre sencillo, nombre de la versión, referencia cultural admitida y clave pública, como se indica a continuación: `ResourceLib, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null` Los recursos se pueden incrustar en cualquier ensamblado al que la aplicación pueda tener acceso. Si especifica un carácter comodín (*) para `assemblyFullName` , el tiempo de ejecución de Entity Framework buscará los recursos en las siguientes ubicaciones, en este orden: 1. el ensamblado que realiza la llamada. 2. los ensamblados a los que se hace referencia. 3. los ensamblados del directorio bin de una aplicación. Si los archivos no están en ninguna de estas ubicaciones, se lanzará una excepción. **Nota:** Cuando se usa el carácter comodín (*), el Entity Framework tiene que buscar en todos los ensamblados los recursos con el nombre correcto. Para mejorar el rendimiento, especifique el nombre de ensamblado en lugar del carácter comodín.
`resourceName`	Nombre del recurso incluido, como AdventureWorksModel. CSDL. Los servicios de metadatos sólo buscarán los archivos o recursos con una de las extensiones siguientes: .csdl, .ssdl o .msl. Si no se especifica `resourceName` , se cargarán todos los recursos de metadatos. Los recursos deberían tener nombres únicos dentro de un ensamblado. Si varios archivos con el mismo nombre se definen en directorios diferentes en el ensamblado, la información de `resourceName` debe incluir la estructura de carpetas antes del nombre del recurso, por ejemplo nombreDeCarpeta.nombreDeArchivo.csdl. `resourceName` no se requiere al especificar un carácter comodín (*) para `assemblyFullName` .

NOTE

Para mejorar el rendimiento, incruste los recursos en el ensamblado que realiza la llamada, excepto en escenarios sin web donde no haya ninguna referencia a los archivos de metadatos y asignaciones subyacentes en el ensamblado que realiza la llamada.

En el ejemplo siguiente se cargan todos los archivos del modelo y de asignación en el ensamblado que realiza la llamada, ensamblados a los que se hace referencia y otros ensamblados en el directorio bin de una aplicación.

```
Metadata=res://*/
```

El ejemplo siguiente carga el archivo model.csdl del ensamblado AdventureWorks y carga los archivos model.ssdl y model.msl desde el directorio predeterminado de la aplicación en ejecución.

```
Metadata=res://AdventureWorks, 1.0.0.0, neutral, a14f3033def15840/model.csdl|model.ssdl|model.msl
```

En el ejemplo siguiente se cargan los tres recursos especificados del ensamblado concreto.

```
Metadata=res://AdventureWorks, 1.0.0.0, neutral, a14f3033def15840/model.csdl|
res://AdventureWorks, 1.0.0.0, neutral, a14f3033def15840/model.ssdl|
res://AdventureWorks, 1.0.0.0, neutral, a14f3033def15840/model.msl
```

En el ejemplo siguiente se cargan todos los recursos incrustados con las extensiones .csdl, .msl y .ssdl del ensamblado.

```
Metadata=res://AdventureWorks, 1.0.0.0, neutral, a14f3033def15840/
```

En el ejemplo siguiente se cargan todos los recursos de la ruta de acceso relativa al archivo más "datadir\metadata \ " desde la ubicación de ensamblado cargada.

```
Metadata=datadir\metadata\
```

En el ejemplo siguiente se cargan todos los recursos en la ruta de acceso del archivo relativa desde la ubicación de ensamblado cargada.

```
Metadata=.\
```

Compatibilidad con la cadena de sustitución || de DataDirectory y el operador raíz de la aplicación web (~)

`DataDirectory` y el operador ~ se utilizan en ConnectionString como parte de las `Metadata` `Provider Connection String` palabras clave y. El elemento EntityConnection reenvía `DataDirectory` y el operador ~ a MetadataWorkspace y al proveedor de almacenamiento, respectivamente.

TÉRMINO	DESCRIPCIÓN

TÉRMINO	DESCRIPCIÓN
`|DataDirectory|`	Se resuelve como una ruta de acceso relativa a archivos de metadatos y una asignación. Se trata del valor que se establece a través del método `AppDomain.SetData("DataDirectory", objValue)`. La `DataDirectory` cadena de sustitución debe estar rodeada por los caracteres de barra vertical y no puede haber ningún espacio en blanco entre su nombre y los caracteres de barra vertical. El nombre de `DataDirectory` no distingue entre mayúsculas y minúsculas. Si un directorio físico denominado "DataDirectory" tiene que pasarse como miembro de la lista de rutas de acceso de metadatos, agregue espacios en blanco a uno de los dos lados del nombre o a ambos. Por ejemplo: `Metadata="DataDirectory1	DataDirectory	DataDirectory2"`. Una aplicación ASP.NET resuelve || de DataDirectory en la <application root> carpeta "/App_Data".
~	Se resuelve como la raíz de la aplicación web. El carácter ~ en una posición inicial siempre se interpreta como el operador raíz de la aplicación web (~), aunque podría representar un subdirectorio local válido. Para hacer referencia a este tipo de subdirectorio local, el usuario debería pasar `./~` explícitamente.

`DataDirectory` y el operador ~ solo se deberían especificar al principio de una ruta de acceso; no se resuelven en ninguna otra posición. Entity Framework intentará resolver `~/data`, pero tratará `/data/~` como una ruta de acceso física.

Una ruta de acceso que comience con `DataDirectory` o con el operador ~ no se puede resolver como una ruta de acceso física fuera de la rama de `DataDirectory` y el operador ~. Por ejemplo, las rutas de acceso siguientes se resolverán: `~`, `~/data`, `~/bin/Model/SqlServer`. Las rutas de acceso siguientes no se resolverán: `~/..`, `~/../other`.

`DataDirectory` y el operador ~ se puede extender para incluir los subdirectorios, de la forma siguiente: `|DataDirectory|\Model`, `~/bin/Model`

La resolución de la cadena de sustitución `DataDirectory` y el operador ~ no es recursiva. Por ejemplo, cuando `DataDirectory` incluye el carácter `~`, se lanza una excepción. De esta forma se evita una recursividad infinita.

Consulte también

- Trabajar con proveedores de datos
- Consideraciones de implementación
- Administrar conexiones y transacciones
- Cadenas de conexión

Proveedores de datos de Entity Framework

20/02/2020 • 2 minutes to read • Edit Online

En esta sección se proporciona información sobre los proveedores de datos que admiten el Entity Framework.

En esta sección

Proveedor de EntityClient para Entity Framework
Describe el proveedor de datos de EntityClient. Este proveedor transforma en un árbol de comandos canónico las consultas realizadas en un modelo de datos. Un proveedor de datos .NET Framework para Entity Framework puede consumir entonces el árbol de comandos.

SqlClient para Entity Framework
Describe el proveedor de datos de .NET Framework que admite el Entity Framework para su uso con una base de datos de SQL Server.

Secciones relacionadas

Entity Framework (SQL Server Compact)
Describe las limitaciones del proveedor y cómo usar el Entity Framework con una base de datos de SQL Server Compact.

Consulte también

- Trabajo con proveedores de datos

Proveedor de EntityClient para Entity Framework

23/10/2019 • 6 minutes to read • Edit Online

El proveedor de EntityClient es un proveedor de datos que usan las aplicaciones de Entity Framework para tener acceso a los datos descritos en un modelo conceptual. Para obtener información sobre los modelos conceptuales, vea modelado y asignación. EntityClient utiliza otros proveedores de datos .NET Framework para tener acceso al origen de datos. Por ejemplo, EntityClient utiliza el Proveedor de datos .NET Framework para SQL Server (SqlClient) al tener acceso a una base de datos de SQL Server. Para obtener información sobre el proveedor SqlClient, vea SqlClient para el Entity Framework. El proveedor de EntityClient se implementa en el espacio de nombres System.Data.EntityClient.

Administrar conexiones

El Entity Framework se basa en los proveedores de datos ADO.net específicos del almacenamiento proporcionando un EntityConnection a un proveedor de datos subyacente y una base de datos relacional. Para construir un EntityConnection objeto, debe hacer referencia a un conjunto de metadatos que contenga la asignación y los modelos necesarios, y también un nombre de proveedor de datos específico del almacenamiento y una cadena de conexión. Una vez EntityConnection que se haya implementado, se puede tener acceso a las entidades a través de las clases generadas a partir del modelo conceptual.

Se puede especificar una cadena de conexión en el archivo app.config.

El espacio de nombres System.Data.EntityClient también incluye la clase EntityConnectionStringBuilder. Esta clase permite que los programadores creen mediante programación cadenas de conexión sintácticamente correctas, y que analicen y recompilen las cadenas de conexión existentes, utilizando las propiedades y los métodos de la clase. Para obtener más información, vea Cómo: Cree una cadenade conexión de EntityConnection.

Crear consultas

El Entity SQL lenguaje es un dialecto de SQL independiente del almacenamiento que trabaja directamente con esquemas de entidades conceptuales y admite conceptos de Entity Data Model como la herencia y las relaciones. La EntityCommand clase se usa para ejecutar un Entity SQL comando en un modelo de entidad. Cuando se crean objetos de EntityCommand, se puede especificar un nombre de procedimiento almacenado o un texto de consulta. El Entity Framework funciona con proveedores de datos específicos del almacenamiento para traducir Entity SQL los genéricos en consultas específicas del almacenamiento. Para obtener más información sobre Entity SQL cómo escribir consultas, consulte Entity SQL Language.

En el ejemplo siguiente se EntityCommand crea un objeto y se asigna Entity SQL un texto de consulta EntityCommand.CommandText a su propiedad. Esta Entity SQL consulta solicita los productos ordenados por el precio de venta del modelo conceptual. El código siguiente no tiene conocimiento alguno del modelo de almacenamiento.

```
EntityCommand cmd = conn.CreateCommand();
cmd.CommandText = @"SELECT VALUE p
 FROM AdventureWorksEntities.Product AS p
 ORDER BY p.ListPrice";
```

Ejecutar consultas

Cuando se ejecuta una consulta, se analiza y se convierte en un árbol de comandos canónico. Todo el procesamiento subsiguiente se realiza en el árbol de comandos. El árbol de comandos es el medio de comunicación entre System.Data.EntityClient y el proveedor de datos de .NET Framework subyacente, System.Data.SqlClientcomo.

EntityDataReader muestra los resultados de ejecutar EntityCommand en un modelo conceptual. Para ejecutar el comando que devuelve el EntityDataReader, llame a ExecuteReader. EntityDataReader implementa IExtendedDataRecord para describir resultados estructurados enriquecidos.

Administrar transacciones

En Entity Framework, hay dos maneras de utilizar las transacciones: automática y explícita. Las transacciones automáticas usan el espacio de nombres System.Transactions y las transacciones explícitas usan la clase EntityTransaction.

Para actualizar los datos que se exponen a través de un modelo conceptual, consulte How to: Administrar transacciones en el Entity Framework.

En esta sección

Procedimientos: Compilación de una cadena de conexión de EntityConnection

Procedimientos: Ejecutar una consulta que devuelve resultados PrimitiveType

Cómo: Ejecutar una consulta que devuelve resultados StructuralType

Cómo: Ejecutar una consulta que devuelve resultados RefType

Procedimientos: Ejecutar una consulta que devuelve tipos complejos

Cómo: Ejecutar una consulta que devuelve colecciones anidadas

Procedimientos: Ejecutar una consulta de Entity SQL con parámetros mediante EntityCommand

Procedimientos: Ejecutar un procedimiento almacenado parametrizado usando EntityCommand

Procedimientos: Ejecutar una consulta polimórfica

Cómo: Navegar por las relaciones con el operador Navigate

Vea también

- Administrar conexiones y transacciones
- ADO.NET Entity Framework
- Referencia del lenguaje

Procedimiento para compilar una cadena de conexión EntityConnection

23/10/2019 • 2 minutes to read • Edit Online

En este tema se muestra un ejemplo para generar una EntityConnection.

Para ejecutar el código de este ejemplo

1. Agregue el modelo AdventureWorks Sales al proyecto y configure el proyecto para usar el Entity Framework. Para obtener más información, vea Cómo: Use el Asistente paraEntity Data Model.
2. En la página de código de la aplicación, agregue las instrucciones `using` siguientes (`Imports` en Visual Basic):

```
using System;
using System.Collections.Generic;
using System.Collections;
using System.Data.Common;
using System.Data;
using System.IO;
using System.Data.SqlClient;
using System.Data.EntityClient;
using System.Data.Metadata.Edm;
```

```
Imports System.Collections.Generic
Imports System.Collections
Imports System.Data.Common
Imports System.Data
Imports System.IO
Imports System.Data.SqlClient
Imports System.Data.EntityClient
Imports System.Data.Metadata.Edm
```

Ejemplo

En el siguiente ejemplo se inicializa el System.Data.SqlClient.SqlConnectionStringBuilder para el proveedor subyacente y, a continuación, se inicializa el objeto System.Data.EntityClient.EntityConnectionStringBuilder y se pasa este objeto al constructor de EntityConnection.

```
// Specify the provider name, server and database.
string providerName = "System.Data.SqlClient";
string serverName = ".";
string databaseName = "AdventureWorks";

// Initialize the connection string builder for the
// underlying provider.
SqlConnectionStringBuilder sqlBuilder =
    new SqlConnectionStringBuilder();

// Set the properties for the data source.
sqlBuilder.DataSource = serverName;
sqlBuilder.InitialCatalog = databaseName;
sqlBuilder.IntegratedSecurity = true;

// Build the SqlConnection connection string.
string providerString = sqlBuilder.ToString();

// Initialize the EntityConnectionStringBuilder.
EntityConnectionStringBuilder entityBuilder =
    new EntityConnectionStringBuilder();

//Set the provider name.
entityBuilder.Provider = providerName;

// Set the provider-specific connection string.
entityBuilder.ProviderConnectionString = providerString;

// Set the Metadata location.
entityBuilder.Metadata = @"res://*/AdventureWorksModel.csdl|
                           res://*/AdventureWorksModel.ssdl|
                           res://*/AdventureWorksModel.msl";
Console.WriteLine(entityBuilder.ToString());

using (EntityConnection conn =
    new EntityConnection(entityBuilder.ToString()))
{
    conn.Open();
    Console.WriteLine("Just testing the connection.");
    conn.Close();
}
```

```
' Specify the provider name, server and database.
Dim providerName As String = "System.Data.SqlClient"
Dim serverName As String = "."
Dim databaseName As String = "AdventureWorks"

' Initialize the connection string builder for the
' underlying provider.
Dim sqlBuilder As New SqlConnectionStringBuilder()

' Set the properties for the data source.
sqlBuilder.DataSource = serverName
sqlBuilder.InitialCatalog = databaseName
sqlBuilder.IntegratedSecurity = True

' Build the SqlConnection connection string.
Dim providerString As String = sqlBuilder.ToString()

' Initialize the EntityConnectionStringBuilder.
Dim entityBuilder As New EntityConnectionStringBuilder()

'Set the provider name.
entityBuilder.Provider = providerName

' Set the provider-specific connection string.
entityBuilder.ProviderConnectionString = providerString

' Set the Metadata location.
entityBuilder.Metadata =
"res://*/AdventureWorksModel.csdl|res://*/AdventureWorksModel.ssdl|res://*/AdventureWorksModel.msl"
Console.WriteLine(entityBuilder.ToString())

Using conn As New EntityConnection(entityBuilder.ToString())
    conn.Open()
    Console.WriteLine("Just testing the connection.")
    conn.Close()
End Using
```

Vea también

- Cómo: Usar EntityConnection con un contexto del objeto
- Proveedor de EntityClient para Entity Framework

Procedimiento para ejecutar una consulta que devuelve resultados PrimitiveType

23/10/2019 • 2 minutes to read • Edit Online

En este tema se muestra cómo ejecutar un comando en un modelo conceptual usando un objeto EntityCommand, y cómo recuperar los resultados de PrimitiveType usando un objeto EntityDataReader.

Para ejecutar el código de este ejemplo

1. Agregue el modelo AdventureWorks Sales al proyecto y configure el proyecto para usar el Entity Framework. Para obtener más información, consulte Cómo Use el Asistente paraEntity Data Model.
2. En la página de código de la aplicación, agregue las instrucciones `using` siguientes (`Imports` en Visual Basic):

```
using System;
using System.Collections.Generic;
using System.Collections;
using System.Data.Common;
using System.Data;
using System.IO;
using System.Data.SqlClient;
using System.Data.EntityClient;
using System.Data.Metadata.Edm;
```

```
Imports System.Collections.Generic
Imports System.Collections
Imports System.Data.Common
Imports System.Data
Imports System.IO
Imports System.Data.SqlClient
Imports System.Data.EntityClient
Imports System.Data.Metadata.Edm
```

Ejemplo

En este ejemplo se ejecuta una consulta que devuelve un resultado PrimitiveType. Si pasa la siguiente consulta como un argumento a la función `ExecutePrimitiveTypeQuery` , la función muestra el precio de venta medio de todos los `Products` :

```
SELECT VALUE AVG(p.ListPrice) FROM AdventureWorksEntities.Products as p
```

Si pasa una consulta parametrizada, como la siguiente, agregue los objetos EntityParameter a la propiedad Parameters en el objeto EntityCommand.

```
CASE WHEN AVG({@score1,@score2,@score3}) < @total THEN TRUE ELSE FALSE END
```

```
static void ExecutePrimitiveTypeQuery(string esqlQuery)
{
    if (esqlQuery.Length == 0)
    {
        Console.WriteLine("The query string is empty.");
        return;
    }

    using (EntityConnection conn =
        new EntityConnection("name=AdventureWorksEntities"))
    {
        conn.Open();

        // Create an EntityCommand.
        using (EntityCommand cmd = conn.CreateCommand())
        {
            cmd.CommandText = esqlQuery;
            // Execute the command.
            using (EntityDataReader rdr =
                cmd.ExecuteReader(CommandBehavior.SequentialAccess))
            {
                // Start reading results.
                while (rdr.Read())
                {
                    IExtendedDataRecord record = rdr as IExtendedDataRecord;
                    // For PrimitiveType
                    // the record contains exactly one field.
                    int fieldIndex = 0;
                    Console.WriteLine("Value: " + record.GetValue(fieldIndex));
                }
            }
        }
        conn.Close();
    }
}
```

```
Private Shared Sub ExecutePrimitiveTypeQuery(ByVal esqlQuery As String)
    If esqlQuery.Length = 0 Then
        Console.WriteLine("The query string is empty.")
        Exit Sub
    End If

    Using conn As New EntityConnection("name=AdventureWorksEntities")
        conn.Open()

        ' Create an EntityCommand.
        Using cmd As EntityCommand = conn.CreateCommand()
            cmd.CommandText = esqlQuery
            ' Execute the command.
            Using rdr As EntityDataReader = cmd.ExecuteReader(CommandBehavior.SequentialAccess)
                ' Start reading results.
                While rdr.Read()
                    Dim record As IExtendedDataRecord = TryCast(rdr, IExtendedDataRecord)
                    ' For PrimitiveType
                    ' the record contains exactly one field.
                    Dim fieldIndex As Integer = 0
                    Console.WriteLine("Value: " & record.GetValue(fieldIndex))
                End While
            End Using
        End Using
        conn.Close()
    End Using
End Sub
```

Vea también

- Referencia de Entity SQL
- Proveedor de EntityClient para Entity Framework

Procedimiento para ejecutar una consulta que devuelve resultados StructuralType

23/10/2019 • 3 minutes to read • Edit Online

En este tema se muestra cómo ejecutar un comando contra un modelo conceptual usando un objeto EntityCommand, y cómo recuperar los resultados de StructuralType usando un EntityDataReader. Las clases EntityType, RowType y ComplexType se derivan de la clase StructuralType.

Para ejecutar el código de este ejemplo

1. Agregue el modelo AdventureWorks Sales al proyecto y configure el proyecto para usar el Entity Framework. Para obtener más información, consulte Cómo Use el Asistente paraEntity Data Model.
2. En la página de código de la aplicación, agregue las instrucciones `using` siguientes (`Imports` en Visual Basic):

```
using System;
using System.Collections.Generic;
using System.Collections;
using System.Data.Common;
using System.Data;
using System.IO;
using System.Data.SqlClient;
using System.Data.EntityClient;
using System.Data.Metadata.Edm;
```

```
Imports System.Collections.Generic
Imports System.Collections
Imports System.Data.Common
Imports System.Data
Imports System.IO
Imports System.Data.SqlClient
Imports System.Data.EntityClient
Imports System.Data.Metadata.Edm
```

Ejemplo

En este ejemplo se ejecuta una consulta que devuelve resultados EntityType. Si pasa la siguiente consulta como un argumento a la función `ExecuteStructuralTypeQuery`, la función muestra detalles sobre los `Products`:

```
SELECT VALUE Product FROM AdventureWorksEntities.Products AS Product
```

Si pasa una consulta parametrizada, como la siguiente, agregue los objetos EntityParameter a la propiedad Parameters en el objeto EntityCommand.

```
SELECT VALUE product FROM AdventureWorksEntities.Products
    AS product where product.ListPrice >= @price
```

```
static void ExecuteStructuralTypeQuery(string esqlQuery)
{
    if (esqlQuery.Length == 0)
    {
        Console.WriteLine("The query string is empty.");
        return;
    }

    using (EntityConnection conn =
        new EntityConnection("name=AdventureWorksEntities"))
    {
        conn.Open();

        // Create an EntityCommand.
        using (EntityCommand cmd = conn.CreateCommand())
        {
            cmd.CommandText = esqlQuery;
            // Execute the command.
            using (EntityDataReader rdr =
                cmd.ExecuteReader(CommandBehavior.SequentialAccess))
            {
                // Start reading results.
                while (rdr.Read())
                {
                    StructuralTypeVisitRecord(rdr as IExtendedDataRecord);
                }
            }
        }
        conn.Close();
    }
}

static void StructuralTypeVisitRecord(IExtendedDataRecord record)
{
    int fieldCount = record.DataRecordInfo.FieldMetadata.Count;
    for (int fieldIndex = 0; fieldIndex < fieldCount; fieldIndex++)
    {
        Console.Write(record.GetName(fieldIndex) + ": ");

        // If the field is flagged as DbNull, the shape of the value is undetermined.
        // An attempt to get such a value may trigger an exception.
        if (record.IsDBNull(fieldIndex) == false)
        {
            BuiltInTypeKind fieldTypeKind = record.DataRecordInfo.FieldMetadata[fieldIndex].
                FieldType.TypeUsage.EdmType.BuiltInTypeKind;
            // The EntityType, ComplexType and RowType are structural types
            // that have members.
            // Read only the PrimitiveType members of this structural type.
            if (fieldTypeKind == BuiltInTypeKind.PrimitiveType)
            {
                // Primitive types are surfaced as plain objects.
                Console.WriteLine(record.GetValue(fieldIndex).ToString());
            }
        }
    }
}
```

```
Private Shared Sub ExecuteStructuralTypeQuery(ByVal esqlQuery As String)
    If esqlQuery.Length = 0 Then
        Console.WriteLine("The query string is empty.")
        Exit Sub
    End If

    Using conn As New EntityConnection("name=AdventureWorksEntities")
        conn.Open()

        ' Create an EntityCommand.
        Using cmd As EntityCommand = conn.CreateCommand()
            cmd.CommandText = esqlQuery
            ' Execute the command.
            Using rdr As EntityDataReader = cmd.ExecuteReader(CommandBehavior.SequentialAccess)
                ' Start reading results.
                While rdr.Read()
                    StructuralTypeVisitRecord(TryCast(rdr, IExtendedDataRecord))
                End While
            End Using
        End Using
        conn.Close()
    End Using
End Sub

Private Shared Sub StructuralTypeVisitRecord(ByVal record As IExtendedDataRecord)
    Dim fieldCount As Integer = record.DataRecordInfo.FieldMetadata.Count
    For fieldIndex As Integer = 0 To fieldCount - 1
        Console.Write(record.GetName(fieldIndex) & ": ")

        ' If the field is flagged as DbNull, the shape of the value is undetermined.
        ' An attempt to get such a value may trigger an exception.
        If record.IsDBNull(fieldIndex) = False Then
            Dim fieldTypeKind As BuiltInTypeKind =
record.DataRecordInfo.FieldMetadata(fieldIndex).FieldType.TypeUsage.EdmType.BuiltInTypeKind
            ' The EntityType, ComplexType and RowType are structural types
            ' that have members.
            ' Read only the PrimitiveType members of this structural type.
            If fieldTypeKind = BuiltInTypeKind.PrimitiveType Then
                ' Primitive types are surfaced as plain objects.
                Console.WriteLine(record.GetValue(fieldIndex).ToString())
            End If
        End If
    Next
End Sub
```

Vea también

- Referencia de Entity SQL
- Proveedor de EntityClient para Entity Framework

Procedimiento para ejecutar una consulta que devuelve resultados RefType

23/10/2019 • 3 minutes to read • Edit Online

En este tema se muestra cómo ejecutar un comando contra un modelo conceptual usando un objeto EntityCommand, y cómo recuperar los resultados de RefType usando un EntityDataReader.

Para ejecutar el código de este ejemplo

1. Agregue el modelo AdventureWorks Sales al proyecto y configure el proyecto para usar el Entity Framework. Para obtener más información, vea Cómo: Use el Asistente paraEntity Data Model.
2. En la página de código de la aplicación, agregue las instrucciones `using` siguientes (`Imports` en Visual Basic):

```
using System;
using System.Collections.Generic;
using System.Collections;
using System.Data.Common;
using System.Data;
using System.IO;
using System.Data.SqlClient;
using System.Data.EntityClient;
using System.Data.Metadata.Edm;
```

```
Imports System.Collections.Generic
Imports System.Collections
Imports System.Data.Common
Imports System.Data
Imports System.IO
Imports System.Data.SqlClient
Imports System.Data.EntityClient
Imports System.Data.Metadata.Edm
```

Ejemplo

En este ejemplo se ejecuta una consulta que devuelve resultados RefType. Si pasa la siguiente consulta como un argumento a la función `ExecuteRefTypeQuery`, la función devuelve una referencia a la entidad:

```
SELECT REF(p) FROM AdventureWorksEntities.Products as p
```

Si pasa una consulta parametrizada, como la siguiente, agregue los objetos EntityParameter a la propiedad Parameters en el objeto EntityCommand.

```
SELECT REF(p) FROM AdventureWorksEntities.Products as p WHERE p.ProductID == @productID
```

```
static public void ExecuteRefTypeQuery(string esqlQuery)
{
    if (esqlQuery.Length == 0)
    {
        Console.WriteLine("The query string is empty.");
        return;
    }

    using (EntityConnection conn =
        new EntityConnection("name=AdventureWorksEntities"))
    {
        conn.Open();

        // Create an EntityCommand.
        using (EntityCommand cmd = conn.CreateCommand())
        {
            cmd.CommandText = esqlQuery;
            // Execute the command.
            using (EntityDataReader rdr =
                cmd.ExecuteReader(CommandBehavior.SequentialAccess))
            {
                // Start reading results.
                while (rdr.Read())
                {
                    RefTypeVisitRecord(rdr as IExtendedDataRecord);
                }
            }
        }
        conn.Close();
    }
}

static void RefTypeVisitRecord(IExtendedDataRecord record)
{
    // For RefType the record contains exactly one field.
    int fieldIndex = 0;

    // If the field is flagged as DbNull, the shape of the value is undetermined.
    // An attempt to get such a value may trigger an exception.
    if (record.IsDBNull(fieldIndex) == false)
    {
        BuiltInTypeKind fieldTypeKind = record.DataRecordInfo.FieldMetadata[fieldIndex].
            FieldType.TypeUsage.EdmType.BuiltInTypeKind;
        //read only fields that contain PrimitiveType
        if (fieldTypeKind == BuiltInTypeKind.RefType)
        {
            // Ref types are surfaced as EntityKey instances.
            // The containing record sees them as atomic.
            EntityKey key = record.GetValue(fieldIndex) as EntityKey;
            // Get the EntitySet name.
            Console.WriteLine("EntitySetName " + key.EntitySetName);
            // Get the Name and the Value information of the EntityKey.
            foreach (EntityKeyMember keyMember in key.EntityKeyValues)
            {
                Console.WriteLine("   Key Name: " + keyMember.Key);
                Console.WriteLine("   Key Value: " + keyMember.Value);
            }
        }
    }
}
```

```
Private Shared Sub ExecuteRefTypeQuery(ByVal esqlQuery As String)
    If esqlQuery.Length = 0 Then
        Console.WriteLine("The query string is empty.")
        Exit Sub
    End If

    Using conn As New EntityConnection("name=AdventureWorksEntities")
        conn.Open()

        ' Create an EntityCommand.
        Using cmd As EntityCommand = conn.CreateCommand()
            cmd.CommandText = esqlQuery
            ' Execute the command.
            Using rdr As EntityDataReader = cmd.ExecuteReader(CommandBehavior.SequentialAccess)
                ' Start reading results.
                While rdr.Read()
                    RefTypeVisitRecord(TryCast(rdr, IExtendedDataRecord))
                End While
            End Using
        End Using
        conn.Close()
    End Using
End Sub

Private Shared Sub RefTypeVisitRecord(ByVal record As IExtendedDataRecord)
    ' For RefType the record contains exactly one field.
    Dim fieldIndex As Integer = 0

    ' If the field is flagged as DbNull, the shape of the value is undetermined.
    ' An attempt to get such a value may trigger an exception.
    If record.IsDBNull(fieldIndex) = False Then
        Dim fieldTypeKind As BuiltInTypeKind =
record.DataRecordInfo.FieldMetadata(fieldIndex).FieldType.TypeUsage.EdmType.BuiltInTypeKind
        'read only fields that contain PrimitiveType
        If fieldTypeKind = BuiltInTypeKind.RefType Then
            ' Ref types are surfaced as EntityKey instances.
            ' The containing record sees them as atomic.
            Dim key As EntityKey = TryCast(record.GetValue(fieldIndex), EntityKey)
            ' Get the EntitySet name.
            Console.WriteLine("EntitySetName " & key.EntitySetName)
            ' Get the Name and the Value information of the EntityKey.
            For Each keyMember As EntityKeyMember In key.EntityKeyValues
                Console.WriteLine(" Key Name: " & keyMember.Key)
                Console.WriteLine(" Key Value: " & keyMember.Value)
            Next
        End If
    End If
End Sub
```

Vea también

- Referencia de Entity SQL
- Proveedor de EntityClient para Entity Framework

Procedimiento para ejecutar una consulta que devuelve tipos complejos

23/10/2019 • 3 minutes to read • Edit Online

En este tema se muestra cómo ejecutar una consulta Entity SQL que devuelve objetos de tipo entidad que contienen una propiedad de un tipo complejo.

Para ejecutar el código de este ejemplo

1. Agregue el modelo AdventureWorks Sales al proyecto y configure el proyecto para usar el Entity Framework. Para obtener más información, consulte Cómo Use el Asistente paraEntity Data Model.

2. En la página de código de la aplicación, agregue las instrucciones `using` siguientes (`Imports` en Visual Basic):

```
using System;
using System.Collections.Generic;
using System.Collections;
using System.Data.Common;
using System.Data;
using System.IO;
using System.Data.SqlClient;
using System.Data.EntityClient;
using System.Data.Metadata.Edm;
```

```
Imports System.Collections.Generic
Imports System.Collections
Imports System.Data.Common
Imports System.Data
Imports System.IO
Imports System.Data.SqlClient
Imports System.Data.EntityClient
Imports System.Data.Metadata.Edm
```

3. Haga doble clic en el archivo. edmx para mostrar el modelo en la ventana Explorador de modelos de Entity Designer. En la superficie de Entity Designer, seleccione `Email` las `Phone` propiedades y del `Contact` tipo de entidad, haga clic con el botón derecho y seleccione **refactorizar en nuevo tipo complejo**.

4. Un nuevo tipo complejo con las propiedades `Email` y `Phone` seleccionadas se agrega al **Explorador de modelos**. Al tipo complejo se le asigna un nombre predeterminado: cambie el nombre del `EmailPhone` tipo a en la ventana **propiedades** . También se agrega una nueva propiedad `ComplexProperty` al tipo de entidad `Contact`. Cambie el nombre de la propiedad a `EmailPhoneComplexType.` .

   Para obtener información acerca de cómo crear y modificar tipos complejos mediante el Asistente para Entity Data Model , consulte How to: Refactorice las propiedades existentes en una propiedad de tipo complejo y cómo: Crear y modificar tiposcomplejos.

Ejemplo

En el ejemplo siguiente se ejecuta una consulta que devuelve una colección `Contact` de objetos y muestra dos propiedades de `Contact` los objetos `ContactID` : y los valores del `EmailPhoneComplexType` tipo complejo.

```
using (EntityConnection conn =
    new EntityConnection("name=AdventureWorksEntities"))
{
    conn.Open();

    string esqlQuery = @"SELECT VALUE contacts FROM
            AdventureWorksEntities.Contacts AS contacts
            WHERE contacts.ContactID == @id";

    // Create an EntityCommand.
    using (EntityCommand cmd = conn.CreateCommand())
    {
        cmd.CommandText = esqlQuery;
        EntityParameter param = new EntityParameter();
        param.ParameterName = "id";
        param.Value = 3;
        cmd.Parameters.Add(param);

        // Execute the command.
        using (EntityDataReader rdr =
            cmd.ExecuteReader(CommandBehavior.SequentialAccess))
        {
            // The result returned by this query contains
            // Address complex Types.
            while (rdr.Read())
            {
                // Display CustomerID
                Console.WriteLine("Contact ID: {0}",
                    rdr["ContactID"]);
                // Display Address information.
                DbDataRecord nestedRecord =
                    rdr["EmailPhoneComplexProperty"] as DbDataRecord;
                Console.WriteLine("Email and Phone Info:");
                for (int i = 0; i < nestedRecord.FieldCount; i++)
                {
                    Console.WriteLine("  " + nestedRecord.GetName(i) +
                        ": " + nestedRecord.GetValue(i));
                }
            }
        }
    }
    conn.Close();
}
```

```
Using conn As New EntityConnection("name=AdventureWorksEntities")
    conn.Open()

    Dim esqlQuery As String = "SELECT VALUE contacts FROM" & _
        " AdventureWorksEntities.Contacts AS contacts WHERE contacts.ContactID == @id"

    ' Create an EntityCommand.
    Using cmd As EntityCommand = conn.CreateCommand()
        cmd.CommandText = esqlQuery
        Dim param As New EntityParameter()
        param.ParameterName = "id"
        param.Value = 3
        cmd.Parameters.Add(param)
        ' Execute the command.
        Using rdr As EntityDataReader = cmd.ExecuteReader(CommandBehavior.SequentialAccess)
            ' The result returned by this query contains
            ' Address complex Types.
            While rdr.Read()
                ' Display CustomerID
                Console.WriteLine("Contact ID: {0}", rdr("ContactID"))
                ' Display Address information.
                Dim nestedRecord As DbDataRecord = TryCast(rdr("EmailPhoneComplexProperty"), DbDataRecord)
                Console.WriteLine("Email and Phone Info:")
                For i As Integer = 0 To nestedRecord.FieldCount - 1
                    Console.WriteLine((" " & nestedRecord.GetName(i) & ": ") + nestedRecord.GetValue(i))
                Next
            End While
        End Using
    End Using
    conn.Close()
End Using
```

Procedimiento para ejecutar una consulta que devuelve colecciones anidadas

23/10/2019 • 2 minutes to read • Edit Online

Aquí se explica cómo ejecutar un comando en un modelo conceptual utilizando un objeto EntityCommand y cómo recuperar los resultados de la colección anidados utilizando EntityDataReader.

Para ejecutar el código de este ejemplo

1. Agregue el modelo AdventureWorks Sales al proyecto y configure el proyecto para usar el Entity Framework. Para obtener más información, consulte Cómo Use el Asistente paraEntity Data Model.
2. En la página de código de la aplicación, agregue las instrucciones `using` siguientes (`Imports` en Visual Basic):

```
using System;
using System.Collections.Generic;
using System.Collections;
using System.Data.Common;
using System.Data;
using System.IO;
using System.Data.SqlClient;
using System.Data.EntityClient;
using System.Data.Metadata.Edm;
```

```
Imports System.Collections.Generic
Imports System.Collections
Imports System.Data.Common
Imports System.Data
Imports System.IO
Imports System.Data.SqlClient
Imports System.Data.EntityClient
Imports System.Data.Metadata.Edm
```

Ejemplo

Una *colección anidada* es una colección que está dentro de otra colección. En el código siguiente se recupera una colección de `Contacts` y las colecciones anidadas de `SalesOrderHeaders` asociadas a cada `Contact`.

```
using (EntityConnection conn =
    new EntityConnection("name=AdventureWorksEntities"))
{
    conn.Open();
    // Create an EntityCommand.
    using (EntityCommand cmd = conn.CreateCommand())
    {
        // Create a nested query.
        string esqlQuery =
            @"Select c.ContactID, c.SalesOrderHeaders
        From AdventureWorksEntities.Contacts as c";

        cmd.CommandText = esqlQuery;
        // Execute the command.
        using (EntityDataReader rdr =
            cmd.ExecuteReader(CommandBehavior.SequentialAccess))
        {
            // The result returned by this query contains
            // ContactID and a nested collection of SalesOrderHeader items.
            // associated with this Contact.
            while (rdr.Read())
            {
                // the first column contains Contact ID.
                Console.WriteLine("Contact ID: {0}", rdr["ContactID"]);

                // The second column contains a collection of SalesOrderHeader
                // items associated with the Contact.
                DbDataReader nestedReader = rdr.GetDataReader(1);
                while (nestedReader.Read())
                {
                    Console.WriteLine("   SalesOrderID: {0} ", nestedReader["SalesOrderID"]);
                    Console.WriteLine("   OrderDate: {0} ", nestedReader["OrderDate"]);
                }
            }
        }
    }
    conn.Close();
}
```

```
Using conn As New EntityConnection("name=AdventureWorksEntities")
    conn.Open()
    ' Create an EntityCommand.
    Using cmd As EntityCommand = conn.CreateCommand()
        ' Create a nested query.
        Dim esqlQuery As String = "Select c.ContactID, c.SalesOrderHeaders" & _
            " From AdventureWorksEntities.Contacts as c"

        cmd.CommandText = esqlQuery
        ' Execute the command.
        Using rdr As EntityDataReader = cmd.ExecuteReader(CommandBehavior.SequentialAccess)
            ' The result returned by this query contains
            ' ContactID and a nested collection of SalesOrderHeader items.
            ' associated with this Contact.
            While rdr.Read()
                ' the first column contains Contact ID.
                Console.WriteLine("Contact ID: {0}", rdr("ContactID"))

                ' The second column contains a collection of SalesOrderHeader
                ' items associated with the Contact.
                Dim nestedReader As DbDataReader = rdr.GetDataReader(1)
                While nestedReader.Read()
                    Console.WriteLine(" SalesOrderID: {0} ", nestedReader("SalesOrderID"))
                    Console.WriteLine(" OrderDate: {0} ", nestedReader("OrderDate"))
                End While
            End While
        End Using
    End Using
    conn.Close()
End Using
```

Vea también

- Proveedor de EntityClient para Entity Framework

Procedimiento para ejecutar una consulta parametrizada de Entity SQL mediante EntityCommand

23/10/2019 • 2 minutes to read • Edit Online

En este tema se muestra cómo ejecutar Entity SQL una consulta que tiene parámetros mediante un EntityCommand objeto.

Para ejecutar el código de este ejemplo

1. Agregue el modelo AdventureWorks Sales al proyecto y configure el proyecto para usar el Entity Framework. Para obtener más información, consulte Cómo Use el Asistente paraEntity Data Model.
2. En la página de código de la aplicación, agregue las instrucciones `using` siguientes (`Imports` en Visual Basic):

```
using System;
using System.Collections.Generic;
using System.Collections;
using System.Data.Common;
using System.Data;
using System.IO;
using System.Data.SqlClient;
using System.Data.EntityClient;
using System.Data.Metadata.Edm;
```

```
Imports System.Collections.Generic
Imports System.Collections
Imports System.Data.Common
Imports System.Data
Imports System.IO
Imports System.Data.SqlClient
Imports System.Data.EntityClient
Imports System.Data.Metadata.Edm
```

Ejemplo

En el ejemplo siguiente se muestra cómo construir una cadena de consulta con dos parámetros. Después crea un EntityCommand, agrega dos parámetros a la colección EntityParameter de ese EntityCommand, y procesa una iteración en la colección de elementos `Contact` .

```
using (EntityConnection conn =
    new EntityConnection("name=AdventureWorksEntities"))
{
    conn.Open();
    // Create a query that takes two parameters.
    string esqlQuery =
        @"SELECT VALUE Contact FROM AdventureWorksEntities.Contacts
                    AS Contact WHERE Contact.LastName = @ln AND
                    Contact.FirstName = @fn";

    using (EntityCommand cmd = new EntityCommand(esqlQuery, conn))
    {
        // Create two parameters and add them to
        // the EntityCommand's Parameters collection
        EntityParameter param1 = new EntityParameter();
        param1.ParameterName = "ln";
        param1.Value = "Adams";
        EntityParameter param2 = new EntityParameter();
        param2.ParameterName = "fn";
        param2.Value = "Frances";

        cmd.Parameters.Add(param1);
        cmd.Parameters.Add(param2);

        using (DbDataReader rdr = cmd.ExecuteReader(CommandBehavior.SequentialAccess))
        {
            // Iterate through the collection of Contact items.
            while (rdr.Read())
            {
                Console.WriteLine(rdr["FirstName"]);
                Console.WriteLine(rdr["LastName"]);
            }
        }
    }
    conn.Close();
}
```

```
Using conn As New EntityConnection("name=AdventureWorksEntities")
    conn.Open()
    ' Create a query that takes two parameters.
    Dim esqlQuery As String = "SELECT VALUE Contact FROM AdventureWorksEntities.Contacts " & _
        " AS Contact WHERE Contact.LastName = @ln AND Contact.FirstName = @fn"

    Using cmd As New EntityCommand(esqlQuery, conn)
        ' Create two parameters and add them to
        ' the EntityCommand's Parameters collection
        Dim param1 As New EntityParameter()
        param1.ParameterName = "ln"
        param1.Value = "Adams"
        Dim param2 As New EntityParameter()
        param2.ParameterName = "fn"
        param2.Value = "Frances"

        cmd.Parameters.Add(param1)
        cmd.Parameters.Add(param2)

        Using rdr As DbDataReader = cmd.ExecuteReader(CommandBehavior.SequentialAccess)
            ' Iterate through the collection of Contact items.
            While rdr.Read()
                Console.WriteLine(rdr("FirstName"))
                Console.WriteLine(rdr("LastName"))
            End While
        End Using
    End Using
    conn.Close()
End Using
```

Vea también

- Cómo: Ejecutar una consulta con parámetros
- Lenguaje Entity SQL

Procedimiento para ejecutar un procedimiento almacenado parametrizado mediante EntityCommand

23/10/2019 • 2 minutes to read • Edit Online

En este tema se muestra cómo ejecutar un procedimiento almacenado parametrizado usando la clase EntityCommand.

Para ejecutar el código de este ejemplo

1. Agregue el modelo School al proyecto y configure el proyecto para que use el Entity Framework. Para obtener más información, consulte Cómo Use el Asistente paraEntity Data Model.
2. En la página de código de la aplicación, agregue las instrucciones `using` siguientes (`Imports` en Visual Basic):

```
using System;
using System.Collections.Generic;
using System.Collections;
using System.Data.Common;
using System.Data;
using System.IO;
using System.Data.SqlClient;
using System.Data.EntityClient;
using System.Data.Metadata.Edm;
```

```
Imports System.Collections.Generic
Imports System.Collections
Imports System.Data.Common
Imports System.Data
Imports System.IO
Imports System.Data.SqlClient
Imports System.Data.EntityClient
Imports System.Data.Metadata.Edm
```

3. Importe el procedimiento almacenado `GetStudentGrades` y especifique entidades `CourseGrade` como tipo de valor devuelto. Para obtener información sobre cómo importar un procedimiento almacenado, consulte cómo: Importar un procedimientoalmacenado.

Ejemplo

El código siguiente ejecuta el procedimiento almacenado `GetStudentGrades` donde `StudentId` es un parámetro necesario. Los resultados los lee después una clase EntityDataReader.

```
using (EntityConnection conn =
    new EntityConnection("name=SchoolEntities"))
{
    conn.Open();
    // Create an EntityCommand.
    using (EntityCommand cmd = conn.CreateCommand())
    {
        cmd.CommandText = "SchoolEntities.GetStudentGrades";
        cmd.CommandType = CommandType.StoredProcedure;
        EntityParameter param = new EntityParameter();
        param.Value = 2;
        param.ParameterName = "StudentID";
        cmd.Parameters.Add(param);

        // Execute the command.
        using (EntityDataReader rdr =
            cmd.ExecuteReader(CommandBehavior.SequentialAccess))
        {
            // Read the results returned by the stored procedure.
            while (rdr.Read())
            {
                Console.WriteLine("ID: {0} Grade: {1}", rdr["StudentID"], rdr["Grade"]);
            }
        }
    }
    conn.Close();
}
```

```
Using conn As New EntityConnection("name=SchoolEntities")
    conn.Open()
    ' Create an EntityCommand.
    Using cmd As EntityCommand = conn.CreateCommand()
        cmd.CommandText = "SchoolEntities.GetStudentGrades"
        cmd.CommandType = CommandType.StoredProcedure
        Dim param As New EntityParameter()
        param.Value = 2
        param.ParameterName = "StudentID"
        cmd.Parameters.Add(param)

        ' Execute the command.
        Using rdr As EntityDataReader = cmd.ExecuteReader(CommandBehavior.SequentialAccess)
            ' Read the results returned by the stored procedure.
            While rdr.Read()
                Console.WriteLine("ID: {0} Grade: {1}", rdr("StudentID"), rdr("Grade"))
            End While
        End Using
    End Using
    conn.Close()
End Using
```

Vea también

- Proveedor de EntityClient para Entity Framework

Procedimiento para ejecutar una consulta polimórfica

23/10/2019 • 2 minutes to read • Edit Online

En este tema se muestra cómo ejecutar una consulta Entity SQL polimórfica mediante el operador de tipo .

Para ejecutar el código de este ejemplo

1. Agregue el modelo School al proyecto y configure el proyecto para que use el Entity Framework. Para obtener más información, consulte Cómo Use el Asistente paraEntity Data Model.
2. En la página de código de la aplicación, agregue las instrucciones `using` siguientes (`Imports` en Visual Basic):

```
using System;
using System.Collections.Generic;
using System.Collections;
using System.Data.Common;
using System.Data;
using System.IO;
using System.Data.SqlClient;
using System.Data.EntityClient;
using System.Data.Metadata.Edm;
```

```
Imports System.Collections.Generic
Imports System.Collections
Imports System.Data.Common
Imports System.Data
Imports System.IO
Imports System.Data.SqlClient
Imports System.Data.EntityClient
Imports System.Data.Metadata.Edm
```

3. Modifique el modelo conceptual para que tenga una herencia de tabla por jerarquía siguiendo los pasos de Tutorial: Asignación de herencia: tabla por jerarquía.

Ejemplo

En el ejemplo siguiente se usa un operador OFTYPE para obtener y mostrar una colección únicamente de `OnsiteCourses` a partir de una colección de `Courses` .

```
using (EntityConnection conn = new EntityConnection("name=SchoolEntities"))
{
    conn.Open();
    // Create a query that specifies to
    // get a collection of only OnsiteCourses.

    string esqlQuery = @"SELECT VAlUE onsiteCourse FROM
        OFTYPE(SchoolEntities.Courses, SchoolModel.OnsiteCourse)
        AS onsiteCourse";
    using (EntityCommand cmd = new EntityCommand(esqlQuery, conn))
    {
        // Execute the command.
        using (DbDataReader rdr = cmd.ExecuteReader(CommandBehavior.SequentialAccess))
        {
            // Start reading.
            while (rdr.Read())
            {
                // Display OnsiteCourse's location.
                Console.WriteLine("CourseID: {0} ", rdr["CourseID"]);
                Console.WriteLine("Location: {0} ", rdr["Location"]);
            }
        }
    }
}
```

```
Using conn As New EntityConnection("name=SchoolEntities")
    conn.Open()
    ' Create a query that specifies to
    ' get a collection of only OnsiteCourses.

    Dim esqlQuery As String = "SELECT VAlUE onsiteCourse FROM " & _
        "OFTYPE(SchoolEntities.Courses, SchoolModel.OnsiteCourse) AS onsiteCourse"
    Using cmd As New EntityCommand(esqlQuery, conn)
        ' Execute the command.
        Using rdr As DbDataReader = cmd.ExecuteReader(CommandBehavior.SequentialAccess)
            ' Start reading.
            While rdr.Read()
                ' Display OnsiteCourse's location.
                Console.WriteLine("CourseID: {0} ", rdr("CourseID"))
                Console.WriteLine("Location: {0} ", rdr("Location"))
            End While
        End Using
    End Using
End Using
```

Vea también

- Proveedor de EntityClient para Entity Framework
- Lenguaje Entity SQL

Procedimiento para navegar por las relaciones con el operador Navigate

23/10/2019 • 2 minutes to read • Edit Online

En este tema se muestra cómo ejecutar un comando contra un modelo conceptual usando un objeto EntityCommand, y cómo recuperar los resultados de RefType usando un EntityDataReader.

Para ejecutar el código de este ejemplo

1. Agregue el modelo AdventureWorks Sales al proyecto y configure el proyecto para usar el Entity Framework. Para obtener más información, consulte Cómo Use el Asistente paraEntity Data Model.
2. En la página de código de la aplicación, agregue las instrucciones `using` siguientes (`Imports` en Visual Basic):

```
using System;
using System.Collections.Generic;
using System.Collections;
using System.Data.Common;
using System.Data;
using System.IO;
using System.Data.SqlClient;
using System.Data.EntityClient;
using System.Data.Metadata.Edm;
```

```
Imports System.Collections.Generic
Imports System.Collections
Imports System.Data.Common
Imports System.Data
Imports System.IO
Imports System.Data.SqlClient
Imports System.Data.EntityClient
Imports System.Data.Metadata.Edm
```

Ejemplo

En el ejemplo siguiente se muestra cómo navegar por Entity SQL las relaciones de mediante el operador Navigate . El `Navigate` operador acepta los parámetros siguientes: una instancia de una entidad, el tipo de relación, el extremo de la relación y el comienzo de la relación. Opcionalmente, solo puede pasar una instancia de una entidad y el tipo de relación al `Navigate` operador.

```
using (EntityConnection conn =
    new EntityConnection("name=AdventureWorksEntities"))
{
    conn.Open();
    // Create an EntityCommand.
    using (EntityCommand cmd = conn.CreateCommand())
    {
        // Create an Entity SQL query.
        string esqlQuery =
            @"SELECT address.AddressID, (SELECT VALUE DEREF(soh) FROM
          NAVIGATE(address, AdventureWorksModel.FK_SalesOrderHeader_Address_BillToAddressID)
          AS soh) FROM AdventureWorksEntities.Addresses AS address";

        cmd.CommandText = esqlQuery;

        // Execute the command.
        using (DbDataReader rdr = cmd.ExecuteReader(CommandBehavior.SequentialAccess))
        {
            // Start reading.
            while (rdr.Read())
            {
                Console.WriteLine(rdr["AddressID"]);
            }
        }
    }
    conn.Close();
}
```

```
Using conn As New EntityConnection("name=AdventureWorksEntities")
    conn.Open()
    ' Create an EntityCommand.
    Using cmd As EntityCommand = conn.CreateCommand()
        ' Create an Entity SQL query.
        Dim esqlQuery As String = "SELECT address.AddressID, (SELECT VALUE DEREF(soh) FROM " & _
            " NAVIGATE(address, AdventureWorksModel.FK_SalesOrderHeader_Address_BillToAddressID) " & _
            " AS soh) FROM AdventureWorksEntities.Addresses AS address"


        cmd.CommandText = esqlQuery

        ' Execute the command.
        Using rdr As DbDataReader = cmd.ExecuteReader(CommandBehavior.SequentialAccess)
            ' Start reading.
            While rdr.Read()
                Console.WriteLine(rdr("AddressID"))
            End While
        End Using
    End Using
    conn.Close()
End Using
```

Vea también

- Proveedor de EntityClient para Entity Framework
- Lenguaje Entity SQL

SqlClient para Entity Framework

23/10/2019 • 3 minutes to read • Edit Online

En esta sección se describe el Proveedor de datos de .NET Framework para SQL Server (SqlClient), el cual permite a Entity Framework trabajar sobre Microsoft SQL Server.

Atributo Provider de Schema

`Provider` es un atributo del elemento `Schema` del lenguaje de definición de esquemas de almacenamiento (SSDL).

Para utilizar SqlClient, asigne la cadena "System.Data.SqlClient" al atributo `Provider` del elemento `Schema`.

Atributo ProviderManifestToken de Schema

`ProviderManifestToken` es un atributo necesario del elemento `Schema` en SSDL. Este token se utiliza para cargar el manifiesto del proveedor en escenarios sin conexión. Para obtener más información `ProviderManifestToken` sobre el atributo, vea Schema (elemento) (SSDL).

SqlClient se puede usar como proveedor de datos para diferentes versiones de SQL Server. Estas versiones tienen capacidades distintas. Por ejemplo, SQL Server 2000 no admite `varchar(max)` los tipos y `nvarchar(max)` que se introdujeron con SQL Server 2005.

SqlCliente genera y acepta los tokens del manifiesto del proveedor siguientes para las diferentes versiones de SQL Server.

SQL SERVER 2000	SQL SERVER 2005	SQL SERVER 2008
2000	2005	2008

> **NOTE**
> A partir de Visual Studio 2010, las herramientas de Entity Data Model de ADO.net no admiten SQL Server 2000.

Nombre del espacio de nombres de proveedor

Todos los proveedores deben especificar un espacio de nombres. Esta propiedad indica a Entity Framework qué prefijo usa el proveedor para estructuras concretas, como los tipos y funciones. El espacio de nombres para los manifiestos del proveedor SqlClient es `SqlServer`. Para obtener más información sobre los espacios de nombres, vea espacios de nombres.

Tipos

El proveedor SqlCliente para Entity Framework proporciona información de asignación entre los tipos del modelo conceptual y los tipos de SQL Server. Para obtener más información, vea SqlClient para Entity Framework.

Funciones

El proveedor de SqlClient para Entity Framework define la lista de funciones admitidas por el proveedor. Para obtener una lista de las funciones admitidas, vea SqlClient para funciones de Entity Framework.

En esta sección

SqlClient para las funciones de Entity Framework

SqlClient para tipos de Entity Framework

Problemas conocidos en SqlClient para Entity Framework

Vea también

- Lenguaje Entity SQL
- Referencia del lenguaje
- Problemas conocidos del proveedor SqlClient para Entity Framework

SqlClient para las funciones de Entity Framework

23/10/2019 • 2 minutes to read • Edit Online

El Proveedor de datos .NET Framework para SQL Server (SqlClient) para el Entity Framework proporciona un conjunto de funciones que permiten realizar cálculos de agregación y matemáticos, así como funciones que efectúan operaciones `System.DateTime` y `string`. Estas funciones están en el espacio de nombres `SQLServer`.

Para obtener una lista de las funciones que deben funcionar con cualquier proveedor, vea funciones canónicas.

Para obtener información sobre cómo se asignan las funciones canónicas a SQL Server funciones, vea modelo conceptual canónico a SQL Server asignación de funciones.

En esta sección

Asignación entre las funciones canónicas del modelo conceptual y las funciones de SQL Server

Funciones de agregado

Funciones de fecha y hora

Funciones matemáticas

Funciones de cadena

Funciones del sistema

Vea también

- Referencia de Entity SQL
- Información general sobre Entity SQL

Asignación entre las funciones canónicas del modelo conceptual y las funciones de SQL Server

23/10/2019 • 5 minutes to read • Edit Online

En este tema se describe cómo se asignan las funciones canónicas del modelo conceptual a las funciones de SQL Server correspondientes.

Funciones de fecha y hora

En la tabla siguiente se describe la asignación de funciones de fecha y hora:

FUNCIONES CANÓNICAS	FUNCIONES DE SQL SERVER
AddDays(expression)	`DATEADD(day, number, date)`
AddHours(expression)	`DATEADD(hour, number, date)`
AddMicroseconds (expresión)	`DATEADD(microsecond, number, date)`
AddMilliseconds(expression)	`DATEADD(millisecond, number, date)`
AddMinutes (expresión)	`DATEADD(minute, number, date)`
AddMonths (expresión)	`DATEADD(month, number, date)`
AddNanoseconds (expresión)	`DATEADD(nanosecond, number, date)`
AddSeconds(expression)	`DATEADD(second, number, date)`
AddYears(expression)	`DATEADD(year, number, date)`
CreateDateTime (año, mes, día, hora, minuto, segundo)	En SQL Server 2000 y SQL Server 2005, se crea un valor con formato `datetime` en el servidor. En SQL Server 2008 y versiones posteriores, se crea un valor `datetime2` en el servidor.
CreateDateTimeOffset (año, mes, día, hora, minuto, segundo, tzoffset)	Se crea un valor con formato `datetimeoffset` en el servidor. No se admite en SQL Server 2000 ni en SQL Server 2005.
CreateTime (hora, minuto, segundo)	Se crea un valor con formato `time` en el servidor. No se admite en SQL Server 2000 ni en SQL Server 2005.
CurrentDateTime()	`SysDateTime()` en SQL Server 2008. `GetDate()` en SQL Server 2000 y SQL Server 2005.

FUNCIONES CANÓNICAS	FUNCIONES DE SQL SERVER
CurrentDateTimeOffset()	`SysDateTimeOffset()` en SQL Server 2008. No se admite en SQL Server 2000 ni en SQL Server 2005.
CurrentUtcDateTime()	`SysUtcDateTime()` en SQL Server 2008. `GetUtcDate()` en SQL Server 2000 y SQL Server 2005.
DayOfYear(expression)	`DatePart(dayofyear, expression)`
Day (expresión)	`DatePart(day, expression)`
DiffDays (startExpression, endExpression)	`DATEDIFF(day, startdate, enddate)`
DiffHours (startExpression, endExpression)	`DATEDIFF(hour, startdate, enddate)`
DiffMicroseconds(startExpression, endExpression)	`DATEDIFF(microsecond, startdate, enddate)`
DiffMilliseconds(startExpression, endExpression)	`DATEDIFF(millisecond, startdate, enddate)`
DiffMinutes (startExpression, endExpression)	`DATEDIFF(minute, startdate, enddate)`
DiffNanoseconds (startExpression, endExpression)	`DATEDIFF(nanosecond, startdate, enddate)`
DiffSeconds(startExpression, endExpression)	`DATEDIFF(second, startdate, enddate)`
DiffYears(startExpression, endExpression)	`DATEDIFF(year, startdate, enddate)`
GetTotalOffsetMinutes(DateTimeOffset)	`DatePart(tzoffset, expression)`
Hour (expresión)	`DatePart(hour, expression)`
Milisegundo (expresión)	`DatePart(millisecond, expression)`
Minute (expresión)	`DatePart(minute, expression)`
Month (expresión)	`DatePart(month, expression)`
Second (expresión)	`DatePart(second, expression)`
TRUNCATE (expresión)	Por SQL Server 2000 y SQL Server 2005, se crea un `datetime` valor con formato truncado en el servidor. En SQL Server 2008 y versiones posteriores, se crea un `datetime2` valor `datetimeoffset` o truncado en el servidor.
Year (expresión)	`DatePart(YEAR, expression)`

Funciones de agregado

En la tabla siguiente se describe la asignación de funciones de agregado:

FUNCIONES CANÓNICAS	FUNCIONES DE SQL SERVER
AVG (expresión)	`AVG(expression)`
BigCount(expression)	`BIGCOUNT(expression)`
Count (expresión)	`COUNT(expression)`
Min (expresión)	`MIN(expression)`
Max (expresión)	`MAX(expression)`
StDev(expression)	`STDEV(expression)`
StDevP(expression)	`STDEVP(expression)`
SUM (expresión)	`SUM(expression)`
Var (expresión)	`VAR(expression)`
VarP (expresión)	`VARP(expression)`

Funciones matemáticas

En la tabla siguiente se describe la asignación de las funciones matemáticas:

FUNCIONES CANÓNICAS	FUNCIONES DE SQL SERVER
ABS (valor)	`ABS(value)`
Ceiling (valor)	`CEILING(value)`
Floor(value)	`FLOOR(value)`
Alimentación (valor)	`POWER(value, exponent)`
Round(value)	`ROUND(value, digits, 0)`
Truncar	`ROUND(value , digits, 1)`

Funciones de cadena

En la tabla siguiente se describe la asignación de las funciones de cadena:

FUNCIONES CANÓNICAS	FUNCIONES DE SQL SERVER
Contains (cadena, destino)	`CHARINDEX(target, string)`
Concat (cadena1, cadena2)	cadena1 + cadena2

FUNCIONES CANÓNICAS	FUNCIONES DE SQL SERVER
EndsWith (cadena, destino)	`CHARINDEX(REVERSE(target), REVERSE(string)) = 1` **Nota**: La `CHARINDEX` función devuelve `false` si el `string` está almacenado en una columna de cadena de longitud `target` fija y es una constante. En este caso, se buscará en toda la cadena, incluyendo los espacios finales de relleno que pueda haber. Una posible solución alternativa es recortar los datos de la cadena de longitud fija antes de pasar esta a la función `EndsWith`, como en el ejemplo siguiente: `EndsWith(TRIM(string), target)`
IndexOf(target, string2)	`CHARINDEX(target, string2)`
Left (cadena1, longitud)	`LEFT(string1, length)`
Length (cadena)	`LEN(string)`
LTrim(string)	`LTRIM(string)`
Right (cadena1, longitud)	`RIGHT (string1, length)`
Trim (cadena)	`LTRIM(RTRIM(string))`
Replace (cadena1, cadena2, String3)	`REPLACE(string1, string2, string3)`
REVERSE (cadena)	`REVERSE (string)`
RTrim(string)	`RTRIM(string)`
StartsWith (cadena, destino)	`CHARINDEX(target, string)`
Substring (cadena, Inicio, longitud)	`SUBSTRING(string, start, length)`
ToLower(string)	`LOWER(string)`
ToUpper (cadena)	`UPPER(string)`

Funciones bit a bit

En la tabla siguiente se describe la asignación de las funciones bit a bit:

FUNCIONES CANÓNICAS	FUNCIONES DE SQL SERVER
BitWiseAnd (valor1, valor2)	valor1 & valor2
BitWiseNot (valor)	~valor
BitWiseOr (valor1, valor2)	valor1 | valor2
BitWiseXor (valor1, valor2)	valor1 ^ valor2

Funciones de agregado (SqlClient para Entity Framework)

21/03/2020 • 4 minutes to read • Edit Online

El Proveedor de datos .NET Framework para SQL Server (SqlClient) proporciona funciones de agregado. Las funciones de agregado realizan cálculos en un conjunto de valores de entrada y devuelven un valor. Estas funciones están en el espacio de nombres SqlServer, que está disponible al utilizar SqlClient. La propiedad del espacio de nombres de un proveedor permite a Entity Framework detectar qué prefijo usa este proveedor para estructuras concretas, como tipos y funciones.

A continuación se muestran las funciones de agregado SqlClient.

AVG(expresión)

Devuelve el promedio de los valores de una colección. Se omiten los valores NULL.

Argumentos

Un `Int32` `Int64` , `Double` , `Decimal` , y .

Valor de devolución

Tipo de `expression` .

Ejemplo

```
SELECT VALUE SqlServer.AVG(p.ListPrice)
FROM AdventureWorksEntities.Products AS p
```

CHECKSUM_AGG (colección)

Devuelve la suma de comprobación de los valores de una colección. Se omiten los valores NULL.

Argumentos

A Collection(`Int32`).

Valor de devolución

Un valor de tipo `Int32` .

Ejemplo

```
SELECT VALUE SqlServer.Checksum_Agg(cast(product.ListPrice AS Int32))
FROM AdventureWorksEntities.Products AS product
WHERE product.ListPrice > cast(@price AS Decimal)
```

COUNT(expresión)

Devuelve el número de elementos de una colección como un valor `Int32` .

Argumentos

Un<T de colección>, donde T es uno de los siguientes tipos:

`Boolean`	`Double`	`DateTime`	`DateTimeOffset`
`Time`	`String`	`Binary`	`Guid` (no devuelto en SQL Server 2000)

Valor de devolución

Un valor de tipo `Int32`.

Ejemplo

```
ANYELEMENT(SELECT VALUE SqlServer.COUNT(product.ProductID)
FROM AdventureWorksEntities.Products AS product
WHERE SqlServer.CEILING(product.ListPrice) ==
SqlServer.FLOOR(product.ListPrice))
```

COUNT_BIG(expresión)

Devuelve el número de elementos de una colección como un valor `bigint`.

Argumentos

Una colección(T), donde T es uno de los siguientes tipos:

`Boolean`	`Double`	`DateTime`	`DateTimeOffset`
`Time`	`String`	`Binary`	`Guid` (no devuelto en SQL Server 2000)

Valor de devolución

Un valor de tipo `Int64`.

Ejemplo

```
ANYELEMENT(SELECT VALUE SqlServer.COUNT_BIG(product.ProductID)
FROM AdventureWorksEntities.Products AS product
WHERE SqlServer.CEILING(product.ListPrice) ==
SqlServer.FLOOR(product.ListPrice))
```

MAX(expresión)

Devuelve el valor máximo de la colección.

Argumentos

Una colección(T), donde T es uno de los siguientes tipos:

`Boolean`	`Double`	`DateTime`	`DateTimeOffset`

`Time`	`String`	`Binary`	

Valor de devolución

Tipo de `expression`.

Ejemplo

```
SELECT VALUE SqlServer.MAX(p.ListPrice)
FROM AdventureWorksEntities.Products AS p
```

MIN(expresión)

Devuelve el valor mínimo de una colección.

Argumentos

Una colección(T), donde T es uno de los siguientes tipos:

`Boolean`	`Double`	`DateTime`	`DateTimeOffset`
`Time`	`String`	`Binary`	

Valor de devolución

Tipo de `expression`.

Ejemplo

```
SELECT VALUE SqlServer.MIN(p.ListPrice)
FROM AdventureWorksEntities.Products AS p
```

STDEV(expresión)

Devuelve la desviación típica estadística de todos los valores de la expresión especificada.

Argumentos

A Collection(`Double`).

Valor de devolución

`Double`.

Ejemplo

```
SELECT VALUE SqlServer.STDEV(product.ListPrice)
FROM AdventureWorksEntities.Products AS product
WHERE product.ListPrice > cast(@price AS Decimal)
```

STDEVP(expresión)

Devuelve la desviación estadística estándar para la población de todos los valores de la expresión especificada.

Argumentos

A Collection(`Double`).

Valor de devolución

`Double`.

Ejemplo

```
SELECT VALUE SqlServer.STDEVP(product.ListPrice)
FROM AdventureWorksEntities.Products AS product
WHERE product.ListPrice > cast(@price AS Decimal)
```

SUM(expresión)

Devuelve la suma de todos los valores de la colección.

Argumentos

Una colección(T) donde T es `Int32` uno `Int64` `Double` de `Decimal` los siguientes tipos: , , , .

Valor de devolución

Tipo de `expression`.

Ejemplo

```
SELECT VALUE SqlServer.SUM(p.ListPrice)
FROM AdventureWorksEntities.Products AS p
```

VAR(expresión)

Devuelve la varianza estadística de todos los valores de la expresión especificada.

Argumentos

A Collection(`Double`).

Valor de devolución

`Double`.

Ejemplo

```
SELECT VALUE SqlServer.VAR(product.ListPrice)
FROM AdventureWorksEntities.Products AS product
WHERE product.ListPrice > cast(@price AS Decimal)
```

VARP(expresión)

Devuelve la varianza estadística de la población para todos los valores de la expresión especificada.

Argumentos

A Collection(`Double`).

Valor de devolución

`Double`.

Ejemplo

```
SELECT VALUE SqlServer.VARP(product.ListPrice)
FROM AdventureWorksEntities.Products AS product
WHERE product.ListPrice > cast(@price AS Decimal)
```

Consulte también

- Funciones de agregado (Transact-SQL)
- Lenguaje Entity SQL
- Funciones canónicas de agregado

Funciones de fecha y hora

20/02/2020 • 7 minutes to read • Edit Online

El Proveedor de datos .NET Framework para SQL Server (SqlClient) proporciona funciones de fecha y hora que realizan operaciones en un valor de entrada `System.DateTime` y devuelven un resultado `string`, numérico o `System.DateTime`. Estas funciones están en el espacio de nombres SqlServer, que está disponible al utilizar SqlClient. La propiedad del espacio de nombres de un proveedor permite a Entity Framework detectar qué prefijo usa este proveedor para estructuras concretas, como tipos y funciones. En la tabla siguiente se muestran las funciones de fecha y hora SqlClient.

FUNCIÓN	DESCRIPCIÓN
`DATEADD(datepart, number, date)`	Devuelve un valor `DateTime` nuevo que se basa en sumar un intervalo a la fecha especificada. **Argumentos** `datepart`: valor de tipo `String` que representa qué parte de la fecha se devuelve como el valor nuevo. `number`: valor de tipo `Int32`, `Int64`, `Decimal` o `Double` que se usa para incrementar el valor de `datepart`. `date:` una expresión que devuelve un `DateTime`, o `DateTimeOffset` o `Time` con precisión = [0-7], o una cadena de caracteres en formato de fecha. **Valor devuelto** Valor de tipo `DateTime`, `DateTimeOffset` o `Time` nuevo con precisión = [0-7]. **Ejemplo** `SqlServer.DATEADD('day', 22, cast('6/9/2006' as DateTime))`

FUNCIÓN	DESCRIPCIÓN

`DATEDIFF(datepart,startdate,enddate)`	Devuelve el número de límites de fecha y hora entre dos fechas especificadas.
	
	Argumentos
	
	`datepart` : `String` que representa la parte de la fecha para calcular la diferencia.
	
	`startdate` : la fecha de comienzo para el cálculo es una expresión que devuelve un valor de tipo `DateTime`, `DateTimeOffset` o `Time` con precisión = [0-7], o una cadena de caracteres en formato de fecha.
	
	`enddate:` una fecha de finalización para el cálculo es una expresión que devuelve un `DateTime`, un `DateTimeOffset` o un valor `Time` con precisión = [0-7], o una cadena de caracteres en formato de fecha.
	
	Valor devuelto
	
	Un valor de tipo `Int32`.
	
	Ejemplo
	
	`SqlServer.DATEDIFF('day', cast('6/9/2006' as DateTime),`
	
	`cast('6/20/2006' as DateTime))`

`DATENAME(datepart, date)`	Devuelve una cadena de caracteres que representa el datepart especificado de la fecha especificada. **Argumentos** `datepart`: valor de tipo `String` que representa qué parte de la fecha se devuelve como el valor nuevo. `date`: expresión que devuelve un valor de tipo `DateTime,``DateTimeOffset` o `Time` con precisión = [0-7], o una cadena de caracteres en un formato de fecha.  **Valor devuelto**  La cadena de caracteres que representa el datepart especificado de la fecha especificada.  **Ejemplo**  `SqlServer.DATENAME('year', cast('6/9/2006' as DateTime))`

FUNCIÓN	DESCRIPCIÓN
`DATEPART(datepart, date)`	Devuelve un número entero que representa el Datepart especificado de la fecha dada. **Argumentos** `datepart`: valor de tipo `String` que representa qué parte de la fecha se devuelve como el valor nuevo. `date`: expresión que devuelve un valor de tipo `DateTime,``DateTimeOffset,` o `Time` con precisión = [0-7], o una cadena de caracteres en un formato de fecha.  **Valor devuelto**  El datepart especificado de la fecha especificada como un valor de tipo `Int32`.  **Ejemplo**  `SqlServer.DATEPART('year', cast('6/9/2006' as DateTime))`
`DAY(date)`	Devuelve el día de la fecha especificada como un entero. **Argumentos** `date`: una expresión de tipo `DateTime` o `DateTimeOffset` con precisión = 0-7. **Valor devuelto** Día de la fecha especificada como un valor de tipo `Int32`. **Ejemplo** `SqlServer.DAY(cast('6/9/2006' as DateTime))`
`GETDATE()`	Genera la fecha y hora actuales en el formato interno de SQL Server para los valores datetime. **Valor devuelto** La fecha y hora actuales del sistema como `DateTime` con una precisión de 3. **Ejemplo** `SqlServer.GETDATE()`
`GETUTCDATE()`	Devuelve el valor datetime en formato de hora universal coordinada (UTC) o del meridiano de Greenwich. **Valor devuelto** El valor `DateTime` con una precisión de 3 en formato UTC. **Ejemplo** `SqlServer.GETUTCDATE()`

FUNCIÓN	DESCRIPCIÓN
`MONTH(date)`	Devuelve el mes de la fecha especificada como un número entero. **Argumentos** `date`: una expresión de tipo `DateTime` o `DateTimeOffset` con precisión = 0-7. **Valor devuelto** Mes de la fecha especificada como un `Int32`. **Ejemplo** `SqlServer.MONTH(cast('6/9/2006' as DateTime))`
`YEAR(date)`	Devuelve el año de la fecha especificada como un número entero. **Argumentos** `date`: una expresión de tipo `DateTime` o `DateTimeOffset` con precisión = 0-7. **Valor devuelto** Año de la fecha especificada como un valor de tipo `Int32`. **Ejemplo** `SqlServer.YEAR(cast('6/9/2006' as DateTime))`
`SYSDATETIME()`	Devuelve un valor `DateTime` con una precisión de 7. **Valor devuelto** Valor `DateTime` con una precisión de 7. **Ejemplo** `SqlServer.SYSDATETIME()`
`SYSUTCDATE()`	Devuelve el valor datetime en formato de hora universal coordinada (UTC) o del meridiano de Greenwich. **Valor devuelto** Valor `DateTime` con una precisión = 7 en formato UTC. **Ejemplo** `SqlServer.SYSUTCDATE()`

FUNCIÓN	DESCRIPCIÓN
`SYSDATETIMEOFFSET()`	Devuelve un valor `DateTimeOffset` con una precisión de 7. **Valor devuelto** Valor `DateTimeOffset` con una precisión de 7 en formato UTC. **Ejemplo** `SqlServer.SYSDATETIMEOFFSET()`

Para obtener más información sobre las funciones de fecha y hora que SqlClient admite, vea tipos de datos y funciones de fecha y hora (Transact-SQL).

Consulte también

- SqlClient para las funciones de Entity Framework

Funciones matemáticas

21/03/2020 • 7 minutes to read • Edit Online

El Proveedor de datos .NET Framework para SQL Server (SqlClient) proporciona funciones matemáticas que realizan cálculos con los valores de entrada que se proporcionan como argumentos y devuelven un resultado numérico. Estas funciones están en el espacio de nombres SqlServer, que está disponible al utilizar SqlClient. La propiedad del espacio de nombres de un proveedor permite a Entity Framework detectar qué prefijo usa este proveedor para estructuras concretas, como tipos y funciones. En la tabla siguiente se describen las funciones matemáticas SqlClient.

ABS (expresión)

Lleva a cabo la función que devuelve el valor absoluto.

Argumentos

`expression` : valor de tipo `Int32` , `Int64` , `Double` o `Decimal` .

Valor de devolución

Valor absoluto de la expresión especificada.

Ejemplo

```
SqlServer.ABS(-2)
```

ACOS(expresión)

Devuelve el valor del arcocoseno de la expresión especificada.

Argumentos

`expression` : un valor `Double` .

Valor de devolución

`Double` .

Ejemplo

```
SqlServer.ACOS(.9)
```

ASIN(expresión)

Devuelve el valor del arcoseno de la expresión especificada.

Argumentos

`expression` : un valor `Double` .

Valor de devolución

`Double` .

Ejemplo

```
SqlServer.ASIN(.9)
```

ATAN(expresión)

Devuelve el valor del arcotangente de la expresión numérica especificada.

Argumentos

`expression`: un valor `Double`.

Valor de devolución

`Double`.

Ejemplo

```
SqlServer.ATAN(9)
```

ATN2(expresión, expresión)

Devuelve el ángulo, en radianes, cuya tangente se encuentra entre las dos expresiones numéricas especificadas.

Argumentos

`expression`: un valor `Double`.

Valor de devolución

`Double`.

Ejemplo

```
SqlServer.ATN2(9, 8)
```

CEILING(expresión)

Convierte la expresión especificada al número entero más pequeño mayor o igual que él.

Argumentos

`expression`: valor de tipo `Int32`, `Int64`, `Double` o `Decimal`.

Valor de devolución

Un `Int32` `Int64`, `Double`, `Decimal`, o .

Ejemplo

```
SELECT VALUE product
FROM AdventureWorksEntities.Products AS product
WHERE product.ListPrice ==
SqlServer.CEILING(product.ListPrice)
```

COS(expresión)

Calcula el coseno trigonométrico del ángulo especificado, en radianes.

Argumentos

`expression`: un valor `Double`.

Valor de devolución

`Double`.

Ejemplo

```
SqlServer.COS(45)
```

COT(expresión)

Calcula la cotangente trigonométrica del ángulo especificado, en radianes.

Argumentos

`expression`: un valor `Double`.

Valor de devolución

`Double`.

Ejemplo

```
SqlServer.COT(60)
```

DEGREES(radianes)

Devuelve el ángulo correspondiente en grados.

Argumentos

`expression`: valor de tipo `Int32`, `Int64`, `Double` o `Decimal`.

Valor de devolución

Un `Int32` `Int64`, `Double`, `Decimal`, o .

Ejemplo

```
SqlServer.DEGREES(3.1)
```

EXP(expresión)

Calcula el valor exponencial de la expresión numérica especificada.

Argumentos

`expression`: un valor `Double`.

Valor de devolución

`Double`.

Ejemplo `SqlServer.EXP(1)`

PLANTA(expresión)

Convierte la expresión especificada al número entero más grande que sea menor o igual que ella.

Argumentos

`expression`: un valor `Double`.

Valor de devolución

`Double`.

Ejemplo

```
SELECT VALUE product
FROM AdventureWorksEntities.Products AS product
WHERE product.ListPrice ==
SqlServer.FLOOR(product.ListPrice)
```

LOG(expresión)

Calcula el logaritmo natural de la expresión `float` especificada.

Argumentos

`expression`: un valor `Double`.

Valor de devolución

`Double`.

Ejemplo

`SqlServer.LOG(100)`

LOG10(expresión)

Devuelve el logaritmo en base 10 de la expresión `Double` especificada.

Argumentos

`expression`: un valor `Double`.

Valor de devolución

`Double`.

Ejemplo

`SqlServer.LOG10(100)`

PI()

Devuelve el valor constante de Pi como un `Double`.

Valor de devolución

`Double`.

Ejemplo

`SqlServer.PI()`

POTENCIA(numeric_expression, power_expression)

Calcula el valor de la expresión especificada elevada a la potencia indicada.

Argumentos

`numeric_expression`	Un `Int32` `Int64` , `Double` , `Decimal` , o .
`power_expression`	A `Double` que representa el poder al `numeric_expression` que elevar el archivo .

Valor de devolución

Valor de la `numeric_expression` especificada a la `power_expression` especificada.

Ejemplo

```
SqlServer.POWER(2,7)
```

RADIANS(expresión)

Convierte los grados en radianes.

Argumentos

`expression` : valor de tipo `Int32` , `Int64` , `Double` o `Decimal` .

Valor de devolución

Un `Int32` `Int64` , `Double` , `Decimal` , o .

Ejemplo

```
SqlServer.RADIANS(360.0)
```

RAND([semilla])

Devuelve un valor aleatorio de 0 a 1.

Argumentos

El valor de `Int32` semilla como un archivo . Si la inicialización no se especifica, el motor de base de datos de SQL Server asigna uno de forma aleatoria. Para un valor de inicialización especificado, el resultado devuelto es siempre el mismo.

Valor de devolución

Valor `Double` aleatorio de 0 a 1.

Ejemplo

```
SqlServer.RAND()
```

ROUND(numeric_expression, length[,function])

Devuelve una expresión numérica, redondeada a la longitud o precisión especificadas.

Argumentos

`numeric_expression`	Un `Int32` `Int64` , `Double` , `Decimal` , o .

`length`	Valor `Int32` que representa la precisión a la que se va a redondear `numeric_expression`. Si `length` es un número positivo, `numeric_expression` se redondea al número de posiciones decimales que especifica `length`. Si `length` es un número negativo, `numeric_expression` se redondea a la izquierda del separador decimal, según se especifica en `length`.
`function`	Opcional. Un `Int32` que representa el tipo de operación que se va a realizar. Cuando se omite la función o tiene `numeric_expression` un valor de 0 (predeterminado), se redondea. Cuando se especifica un valor `numeric_expression` distinto de 0, se trunca.

Valor de devolución

Valor de la `numeric_expression` especificada a la `power_expression` especificada.

Ejemplo

```
SqlServer.ROUND(748.58, -3)
```

SIGN(expresión)

Devuelve el signo positivo (+1), cero (0) o negativo (-1) de la expresión especificada.

Argumentos

`expression`: `Int32`, `Int64`, `Double` o `Decimal`

Valor de devolución

Un `Int32` `Int64`, `Double`, `Decimal`, o .

Ejemplo

```
SqlServer.SIGN(-10)
```

SIN(expresión)

Calcula el seno trigonométrico de un ángulo especificado, en radianes, y devuelve una expresión de tipo `Double`.

Argumentos

`expression`: un valor `Double`.

Valor de devolución

`Double`.

Ejemplo `SqlServer.SIN(20)`

SQRT(expresión)

Devuelve la raíz cuadrada de la expresión especificada.

Argumentos

`expression`: un valor `Double`.

Valor de devolución

`Double`.

Ejemplo `SqlServer.SQRT(3600)`

SQUARE(expresión)

Devuelve la raíz cuadrada de la expresión especificada.

Argumentos

`expression`: un valor `Double`.

Valor de devolución

`Double`.

Ejemplo

`SqlServer.SQUARE(25)`

TAN(expresión)

Calcula la tangente de una expresión especificada.

Argumentos

`expression`: `Double`

Valor de devolución

`Double`

Ejemplo

`SqlServer.TAN(45.0)`

Consulte también

- Funciones matemáticas (Transact-SQL)
- SqlClient para las funciones de Entity Framework

Funciones de cadena

21/03/2020 • 13 minutes to read • Edit Online

El proveedor de datos .NET Framework para SQL Server (SqlClient) proporciona funciones de `String` que realizan operaciones en una `String` de entrada y devuelven una `String` o un resultado con un valor numérico. Estas funciones están en el espacio de nombres SqlServer, que está disponible al utilizar SqlClient. La propiedad del espacio de nombres de un proveedor permite a Entity Framework detectar qué prefijo usa este proveedor para estructuras concretas, como tipos y funciones.

En la tabla siguiente se muestran las funciones `String` de SqlClient.

FUNCIÓN	DESCRIPCIÓN
`ASCII(expression)`	Devuelve el valor de código ASCII del carácter situado más a la izquierda de una expresión de cadena. **Argumentos** `expression`: cualquier expresión válida de un tipo `String` ASCII. **Valor de devolución** Un valor de tipo `Int32`. **Ejemplo** `SqlServer.ASCII('A')`
`CHAR(expression)`	Convierte un código `Int32` en una cadena ASCII. **Argumentos** `expression`: valor de tipo `Int32`. **Valor de devolución** Valor de tipo `String` ASCII. **Ejemplo** `SqlServer.char(97)`

FUNCIÓN	DESCRIPCIÓN
`CHARINDEX(expression1, expression2 [, start_location])`	Devuelve la posición inicial de la expresión especificada en una cadena de caracteres. **Argumentos** `expression1`: expresión que contiene la secuencia de caracteres que se va a buscar. La expresión puede ser de un tipo String (ASCII o Unicode) o Binary. `expression2`: expresión, que normalmente es una columna, en la que se encuentra la secuencia especificada. La expresión puede ser de un tipo String (ASCII o Unicode) o Binary. `start_location`:(Opcional) un Int64 (no devuelto en SQL Server 2000) o Int32 que representa la posición del carácter para empezar a buscar expression1 en expression2. Si no se especifica start_location, es un número negativo o es igual a cero, la búsqueda comienza al principio de expression2. **Valor de devolución** Un valor de tipo `Int32`. **Ejemplo** `SqlServer.CHARINDEX('h', 'habcdefgh', 2)`

`DIFFERENCE(expression, expression)`	Compara los valores de `SOUNDEX` de dos cadenas y evalúa la similitud entre ambas.
	
	Argumentos
	
	Tipo `String` Unicode o ASCII. `expression` puede ser una constante, una variable o una columna.
	
	Valor de devolución
	
	Devuelve un valor de tipo `Int32` que representa la diferencia entre los valores de SOUNDEX de dos expresiones de caracteres. El intervalo está comprendido entre 0 y 4. El valor 0 indica una similitud escasa o inexistente, y el valor 4 indica una elevada similitud o que los valores son iguales.
	
	Ejemplo
	
	`// The following example returns a DIFFERENCE value of 4,`
	
	`//the least possible difference or the best match.`
	
	`SqlServer.DIFFERENCE('Green','Greene');`

FUNCIÓN	DESCRIPCIÓN
`LEFT(expression, count)`	Devuelve la parte izquierda de una cadena de caracteres con el número de caracteres especificado. **Argumentos** `expression`: tipo String Unicode o ASCII. Use la función CAST para convertir character_expression explícitamente. `count`: valor de tipo `Int64` (no se devuelve en SQL Server 2000) o `Int32` que especifica cuántos caracteres de character_expression se devolverán. **Valor de devolución** Valor `String` Unicode o ASCII. **Ejemplo** `SqlServer.LEFT('SQL Server', 4)`
`LEN(expression)`	Devuelve el número de caracteres de la expresión de cadena especificada, excluidos los espacios en blanco finales. **Argumentos** `expression`: expresión de un tipo `String` (ASCII o Unicode) o un tipo `Binary`. **Valor de devolución** Un valor de tipo `Int32`. **Ejemplo** `SqlServer.LEN('abcd')`
`LOWER(expression)`	Devuelve una expresión de `String` después de convertir a minúsculas los datos de caracteres en mayúsculas. **Argumentos** `expression`: cualquier expresión válida del tipo `String`. **Valor de devolución** `String`. **Ejemplo** `SqlServer.LOWER('AbB')`

FUNCIÓN	DESCRIPCIÓN
`LTRIM(expression)`	Devuelve una expresión `String` tras quitar los espacios iniciales en blanco. **Argumentos** `expression`: cualquier expresión válida del tipo `String`. **Valor de devolución** `String`. **Ejemplo** `SqlServer.LTRIM(' d')`
`NCHAR(expression)`	Devuelve el valor de tipo `String` Unicode correspondiente al código entero dado, tal como se define en el estándar Unicode. **Argumentos** `expression`: valor de tipo `Int32`. **Valor de devolución** Valor de tipo `String` Unicode. **Ejemplo** `SqlServer.NCHAR(65)`
`PATINDEX('%pattern%', expression)`	Devuelve la posición inicial de la primera aparición de un patrón en una expresión `String` especificada. **Argumentos** `'%pattern%'`: valor de tipo `String` ASCII o Unicode. Se pueden utilizar caracteres comodín; no obstante, el carácter % debe ir delante y detrás del patrón (excepto cuando se busque el primer o último carácter). `expression`: cadena de tipo `String` ASCII o Unicode en la que buscar el patrón especificado. **Valor de devolución** Un valor de tipo `Int32`. **Ejemplo** `SqlServer.PATINDEX('abc', 'ab')`

FUNCIÓN	DESCRIPCIÓN
`QUOTENAME('char_string' [, 'quote_char'])`	Devuelve un valor de tipo `String` Unicode con los delimitadores agregados para convertirla en un identificador delimitado válido de SQL Server 2005. **Argumentos** `char_string`: valor `String` Unicode. `quote_char`: cadena de un solo carácter que se utiliza como delimitador. Puede ser una comilla simple ('), un corchete izquierdo o derecho ([ ]) o una comilla doble ("). Si no se especifica `quote_char`, se usarán corchetes. **Valor de devolución** Valor de tipo `String` Unicode. **Ejemplo** `SqlServer.QUOTENAME('abc[]def')`
`REPLACE(expression1, expression2, expression3)`	Reemplaza una expresión de carácter por otra expresión de carácter. **Argumentos** `expression1`: expresión de cadena que se va a buscar. `expression1` puede ser un tipo de cadena Unicode o ASCII. `expression2`:La subcadena que se va a encontrar. `expression2` puede ser un tipo de cadena Unicode o ASCII. `expression3`: cadena de reemplazo. `expression3` puede ser un tipo de cadena Unicode o ASCII. **Ejemplo** `SqlServer.REPLACE('aabbcc', 'bc', 'zz')`
`REPLICATE(char_expression, int_expression)`	Repite una expresión de carácter un número especificado de veces. **Argumentos** `char_expression`: tipo `String` Unicode o ASCII. `int_expression`: `Int64` (no se admite en SQL Server 2000) o `Int32`. **Valor de devolución** Tipo `String` Unicode o ASCII. **Ejemplo** `SqlServer.REPLICATE('aa',2)`

FUNCIÓN	DESCRIPCIÓN
`REVERSE(expression)`	Devuelve un valor de tipo String Unicode o ASCII con las posiciones de los caracteres invertidas con respecto a la cadena de entrada. **Argumentos** `expression`: tipo `String` Unicode o ASCII. **Valor de devolución** Tipo `String` Unicode o ASCII. **Ejemplo** `SqlServer.REVERSE('abcd')`
`RIGHT(char_expression, count)`	Devuelve la parte derecha de una cadena de caracteres con el número de caracteres especificado. **Argumentos** `char_expression`:Un tipo de cadena Unicode o ASCII. Use la función CAST para convertir character_expression explícitamente. `count`: valor de tipo `Int64` (no se devuelve en SQL Server 2000) o `Int32` que especifica cuántos caracteres de character_expression se devolverán. **Valor de devolución** Un tipo `String` ASCII. **Ejemplo** `SqlServer.RIGHT('SQL Server', 6)`
`RTRIM(expression)`	Devuelve un valor de tipo String Unicode o ASCII después de quitar los espacios finales. **Argumentos** `expression`: tipo `String` Unicode o ASCII. **Valor de devolución** Tipo `String` Unicode o ASCII. **Ejemplo** `SqlServer.RTRIM(' d e ')`

FUNCIÓN	DESCRIPCIÓN

`SOUNDEX(expression)`	Devuelve un código de cuatro caracteres (SOUNDEX) para evaluar la similitud de dos cadenas. **Argumentos**
	
	`expression`: tipo String Unicode o ASCII.
	
	Valor de devolución
	
	Valor de tipo `String` ASCII. Un código de cuatro caracteres (SOUNDEX) es una cadena que evalúa la semejanza de dos cadenas.
	
	Ejemplo
	
	`Select SqlServer.SOUNDEX('Smith'), SqlServer.SOUNDEX('Smythe') FROM {1}`
	
	Devuelve
	
	`----- ----- S530 S530`

`SPACE(int_expression)`	Devuelve un valor de tipo `String` ASCII de espacios repetidos. **Argumentos** `int_expression`: valor de tipo `Int64` (no se devuelve en SQL Server 2000) o `Int32` que indica el número de espacios. **Valor de devolución** Valor de tipo `String` ASCII. **Ejemplo** `SqlServer.SPACE(2)`

FUNCIÓN	DESCRIPCIÓN
`STR(float_expression [, length [, decimal]])`	Devuelve un valor `String` ASCII convertido a partir de datos numéricos. **Argumentos** `float _expression`: expresión de un tipo de datos (`Double`) numérico aproximado con un separador decimal. `length`: (opcional) valor de tipo `Int32` que representa la longitud total. Ésta incluye el separador decimal, el signo, los dígitos y los espacios. El valor predeterminado es 10. `decimal`:(opcional) `Int32` Un que representa el número de lugares a la derecha del punto decimal. decimal debe ser menor o igual que 16. Si decimal es mayor que 16, el resultado se trunca a dieciséis lugares a la derecha del separador decimal. **Valor de devolución** Valor de tipo `String` ASCII. **Ejemplo** `SqlServer.STR(212.0)`
`STUFF(str_expression, start, length, str_expression_to_insert)`	Elimina una cantidad especificada de caracteres e inserta otro conjunto de caracteres a partir del punto inicial especificado de una expresión de cadena. **Argumentos** `str_expression`: valor `String` Unicode o ASCII. `start:` Un `Int64` valor (no devuelto en SQL `Int32` Server 2000) o un valor que especifica la ubicación para iniciar la eliminación y la inserción. `length`: valor `Int64` (no se devuelve en SQL Server 2000) o `Int32` que especifica el número de caracteres que se van a eliminar. `str_expression_to_insert`: valor `String` Unicode o ASCII. **Valor de devolución** Valor `String` Unicode o ASCII. **Ejemplo** `SqlServer.STUFF('abcd', 2, 2, 'zz')`

FUNCIÓN	DESCRIPCIÓN
`SUBSTRING(str_expression, start, length)`	Devuelve parte de una expresión `String`. **Argumentos** `str_expression`: expresión de un tipo `String` (ASCII o Unicode) o un tipo `Binary`. `start`: valor de tipo `Int64` (no se devuelve en SQL Server 2000) o `Int32` que especifica dónde comienza la subcadena. 1 se refiere al primer carácter de la cadena. `length`: valor `Int64` (no se devuelve en SQL Server 2000) o `Int32` que especifica cuántos caracteres de la expresión se devolverán. **Valor de devolución** Valor de tipo `String` (ASCII o Unicode) o `Binary`. **Ejemplo** `SqlServer.SUBSTRING('abcd', 2, 2)`
`UNICODE(expression)`	Devuelve el valor entero, según la definición del estándar Unicode, para el primer carácter de la expresión de entrada. **Argumentos** `expression`: valor `String` Unicode. **Valor de devolución** Un valor de tipo `Int32`. **Ejemplo** `SqlServer.UNICODE('a')`
`UPPER(expression)`	Devuelve una expresión `String` después de convertir a mayúsculas los datos de caracteres que están en minúsculas. **Argumentos** `expression`: expresión de un tipo String ASCII o Unicode. **Valor de devolución** Tipo `String` ASCII o Unicode. **Ejemplo** `SqlServer.UPPER('AbB')`

Para obtener más `String` información acerca de las funciones que admite SqlClient, vea Funciones de cadena (Transact-SQL).

Consulte también

- SqlClient para las funciones de Entity Framework
- Problemas conocidos en SqlClient para Entity Framework

Funciones del sistema

20/02/2020 • 3 minutes to read • Edit Online

El Proveedor de datos .NET Framework para SQL Server (SqlClient) proporciona las funciones del sistema siguientes:

FUNCIÓN	DESCRIPCIÓN
`CHECKSUM ( value , [ value , [ value ]] )`	Devuelve el valor de suma. `CHECKSUM` se ha pensado para utilizarlo en la compilación de índices hash. **Argumentos** `value` : un `Boolean` , `Byte` , `Int16` , `Int32` , `Int64` , `Single` , `Decimal` , `Double` , `DateTime` , `String` , `Binary` o `Guid` . Puede especificar uno, dos o tres valores. **Valor devuelto** Valor absoluto de la expresión especificada. **Ejemplo** `SqlServer.CHECKSUM(10,100,1000.0)`
`CURRENT_TIMESTAMP ()`	Genera la fecha actual y la hora en el formato interno de SQL Server para los valores `DateTime` con una precisión de 7 en SQL Server 2008 y una precisión de 3 en SQL Server 2005. **Valor devuelto** La fecha y la hora actuales del sistema como un `DateTime` . **Ejemplo** `SqlServer.CURRENT_TIMESTAMP()`
`CURRENT_ USER ()`	Devuelve el nombre del usuario actual. **Valor devuelto** Valor de tipo `String` ASCII. **Ejemplo** `SqlServer.CURRENT_USER()`

FUNCIÓN	DESCRIPCIÓN

`DATALENGTH` `(` `expression` `)`	Devuelve el número de bytes utilizados para representar cualquier expresión.
	
	Argumentos
	
	`expression` : un `Boolean` , `Byte` , `Int16` , `Int32` , `Int64` , `Single` , `Decimal` , `Double` , `DateTime` , `Time` , `DateTimeOffset` , `String` , `Binary` o `Guid` .
	
	Valor devuelto
	
	Tamaño de las propiedades en forma de un valor `Int32` .
	
	Ejemplo
	
	`SELECT VALUE SqlServer.DATALENGTH(P.Name)FROM`
	
	`AdventureWorksEntities.Product AS P`

`HOST_NAME()`	Devuelve el nombre de la estación de trabajo. **Valor devuelto** Valor de tipo `String` Unicode. **Ejemplo** `SqlServer.HOST_NAME()`
`ISDATE(` `expression` `)`	Determina si una expresión de entrada es una fecha válida. **Argumentos** `expression` : un `Boolean` , `Byte` , `Int16` , `Int32` , `Int64` , `Single` , `Decimal` , `Double` , `DateTime` , `Time` , `DateTimeOffset` , `String` , `Binary` o `Guid` . **Valor devuelto** Un valor de tipo `Int32` . Uno (1) si la expresión de entrada es una fecha válida. De lo contrario, es cero (0). **Ejemplo** `SqlServer.ISDATE('1/1/2006')`

FUNCIÓN	DESCRIPCIÓN
`ISNUMERIC( expression )`	Determina si una expresión es un tipo numérico válido. **Argumentos** `expression`: un `Boolean`, `Byte`, `Int16`, `Int32`, `Int64`, `Single`, `Decimal`, `Double`, `DateTime`, `Time`, `DateTimeOffset`, `String`, `Binary` o `Guid`. **Valor devuelto** Un valor de tipo `Int32`. Uno (1) si la expresión de entrada es una fecha válida. De lo contrario, es cero (0). **Ejemplo** `SqlServer.ISNUMERIC('21')`
`NEWID()`	Crea un valor único de tipo Guid. **Valor devuelto** `Guid`. **Ejemplo** `SqlServer.NEWID()`
`USER_NAME( id )`	Devuelve un nombre de usuario de base de datos a partir de un número de identificación especificado. **Argumentos** `expression`: número de identificación `Int32` asociado al usuario de una base de datos. **Valor devuelto** Valor de tipo `String` Unicode. **Ejemplo** `SqlServer.USER_NAME(0)`

Para obtener más información sobre las funciones de `String` que SqlClient admite, consulte funciones de cadena (Transact-SQL).

Consulte también

- Lenguaje Entity SQL
- SqlClient para las funciones de Entity Framework

SqlClient para tipos de Entity Framework

08/11/2019 • 7 minutes to read • Edit Online

El archivo del manifiesto del proveedor correspondiente al Proveedor de datos .NET Framework para SQL Server (SqlClient) incluye la lista de tipos primitivos del proveedor, las facetas de cada tipo, las asignaciones entre los tipos primitivos de los modelos conceptual y de almacenamiento, y las reglas de conversión y de promoción entre los tipos primitivos de los modelos conceptual y de almacenamiento.

En la tabla siguiente se describen los tipos para las bases de datos SQL Server 2008, SQL Server 2005 y SQL Server 2000 y cómo se asignan estos tipos a los tipos de modelos conceptuales. Algunos tipos nuevos introducidos en las últimas versiones de SQL Server no se admiten en las versiones más antiguas. Estos tipos se indican en la tabla siguiente.

TIPO DE PROVEEDOR NAME	TIPO DE PROVEEDOR ATRIBUTOS	EDMSIMPLETYPE NAME	FACETS
`bit`	no disponible	`Edm.Boolean`	no disponible
`tinyint`	no disponible	`Edm.Byte`	no disponible
`smallint`	no disponible	`Edm.Int16`	no disponible
`int`	no disponible	`Edm.Int32`	no disponible
`bigint`	no disponible	`Edm.Int64`	no disponible
`float`	no disponible	`Edm.Double`	no disponible
`real`	no disponible	`Edm.Double`	no disponible
`decimal`	no disponible	`Edm.Decimal`	Precisión -Mínimo: 1 -Máximo: 38 -Valor predeterminado: 18 -Constante: false Escala -Mínimo: 0 -Máximo: 38 -Valor predeterminado: 0 -Constante: false

TIPO DE PROVEEDOR NAME	TIPO DE PROVEEDOR ATRIBUTOS	EDMSIMPLETYPE NAME	FACETS
`numeric`	no disponible	`Edm.Decimal`	Precisión -Mínimo: 1 -Máximo: 38 -Valor predeterminado: 18 -Constante: false Escala -Mínimo: 0 -Máximo: 38 -Valor predeterminado: 0 -Constante: false
`smallmoney`	no disponible	`Edm.Decimal`	Precisión -Valor predeterminado: 10 -Constant: true Escala -Valor predeterminado: 4 -Constant: true
`money`	no disponible	`Edm.Decimal`	Precisión -Valor predeterminado: 19 -Constant: true Escala -Valor predeterminado: 4 -Constant: true

TIPO DE PROVEEDOR NAME	TIPO DE PROVEEDOR ATRIBUTOS	EDMSIMPLETYPE NAME	FACETS
`binary`	no disponible	`Edm.Binary`	Longitud -Mínimo: 1 -Máximo: 8000 -Valor predeterminado: 8000 -Constante: false FixedLength -Default: true -Constant: true
`varbinary`	no disponible	`Edm.Binary`	Longitud -Mínimo: 1 -Máximo: 8000 -Valor predeterminado: 8000 -Constante: false FixedLength -Valor predeterminado: false -Constant: true
`varbinary(max)` Nota: este tipo no se admite en SQL Server 2000.	no disponible	`Edm.Binary`	Longitud -Valor predeterminado: 214748364780 -Constant: true FixedLength -Valor predeterminado: false -Constant: true

TIPO DE PROVEEDOR NAME	TIPO DE PROVEEDOR ATRIBUTOS	EDMSIMPLETYPE NAME	FACETS
`image`	no disponible	`Edm.Binary`	Longitud -Valor predeterminado: 2147483647 -Constant: true FixedLength -Valor predeterminado: false -Constant: true
`timestamp`	no disponible	`Edm.Binary`	Longitud -Valor predeterminado: 8 -Constant: true FixedLength -Default: true -Constant: true
`rowversion`	no disponible	`Edm.Binary`	Longitud -Valor predeterminado: 8 -Constant: true FixedLength -Default: true -Constant: true
`smalldatetime`	no disponible	`Edm.DateTime`	Precisión -Valor predeterminado: 0 -Constant: true
`datetime`	no disponible	`Edm.DateTime`	Precisión -Valor predeterminado: 3 -Constant: true
`date` Nota: este tipo no se admite en SQL Server 2005 y SQL Server 2000.	no disponible	`Edm.DateTime`	Precisión -Valor predeterminado: 0 -Constante: false

TIPO DE PROVEEDOR NAME	TIPO DE PROVEEDOR ATRIBUTOS	EDMSIMPLETYPE NAME	FACETS
`time` Nota: este tipo no se admite en SQL Server 2005 y SQL Server 2000.	no disponible	`Edm.Time`	Precisión -Valor predeterminado: 7 -Constante: false
`datetime2` Nota: este tipo no se admite en SQL Server 2005 y SQL Server 2000.	no disponible	`Edm.DateTime`	Precisión -Valor predeterminado: 7 -Constante: false
`datetimeoffset` Nota: este tipo no se admite en SQL Server 2005 y SQL Server 2000.	no disponible	`Edm.DateTimeOffset`	Precisión -Valor predeterminado: 7 -Constante: false
`nvarchar` Nota: este tipo no se admite en SQL Server 2000.	no disponible	`Edm.String`	Longitud -Mínimo: 1 -Máximo: 4000 -Valor predeterminado: 4000 -Constante: false Unicode: -Default: true -Constant: true FixedLength -Valor predeterminado: false -Constant: true

TIPO DE PROVEEDOR NAME	TIPO DE PROVEEDOR ATRIBUTOS	EDMSIMPLETYPE NAME	FACETS
`varchar` Nota: este tipo no se admite en SQL Server 2000.	no disponible	`Edm.String`	Longitud -Mínimo: 1 -Máximo: 8000 -Valor predeterminado: 8000 -Constante: false Unicode: -Valor predeterminado: false -Constant: true FixedLength -Valor predeterminado: false -Constant: true
`char`	no disponible	`Edm.String`	Longitud -Mínimo: 1 -Máximo: 8000 -Valor predeterminado: 8000 -Constante: false Unicode: -Valor predeterminado: false -Constant: true FixedLength -Default: true -Constant: true

TIPO DE PROVEEDOR NAME	TIPO DE PROVEEDOR ATRIBUTOS	EDMSIMPLETYPE NAME	FACETS
`nchar`	no disponible	`Edm.String`	Longitud -Mínimo: 1 -Máximo: 4000 -Valor predeterminado: 4000 -Constante: false Unicode: -Default: true -Constant: true FixedLength -Default: true -Constant: true
`varchar` (`max`)	no disponible	`Edm.String`	Longitud -Valor predeterminado: 2147483647 -Constant: true Unicode: -Valor predeterminado: false -Constant: true FixedLength -Valor predeterminado: false -Constant: true

TIPO DE PROVEEDOR NAME	TIPO DE PROVEEDOR ATRIBUTOS	EDMSIMPLETYPE NAME	FACETS
`nvarchar` (`max`)	no disponible	`Edm.String`	Longitud -Valor predeterminado: 1073741823 -Constant: true Unicode: -Default: true -Constant: true FixedLength -Valor predeterminado: false -Constant: true
`ntext`	Igual comparable: false Orden comparable: false	`Edm.String`	Longitud -Valor predeterminado: 1073741823 -Constant: true Unicode: -Valor predeterminado: false -Constant: true FixedLength -Valor predeterminado: false -Constant: true
`text`	Igual comparable: false Orden comparable: false	`Edm.String`	Longitud -Valor predeterminado: 2147483647 -Constant: true Unicode: -Valor predeterminado: false -Constant: true FixedLength -Valor predeterminado: false -Constant: true

TIPO DE PROVEEDOR NAME	TIPO DE PROVEEDOR ATRIBUTOS	EDMSIMPLETYPE NAME	FACETS
`Unique` `identifier`	Igual comparable: true Orden comparable: true	`Edm.Guid`	no disponible
`xml`	Igual comparable: false Orden comparable: false	`Edm.String`	Longitud -Valor predeterminado: 1073741823 -Constant: true Unicode: -Default: true -Constant: true FixedLength -Valor predeterminado: false -Constant: true

Vea también

- Especificaciones CSDL, SSDL MSL

Problemas conocidos en SqlClient para Entity Framework

20/02/2020 • 7 minutes to read • Edit Online

En esta sección se describen problemas conocidos relacionados con el proveedor de datos .NET Framework para SQL Server (SqlClient).

Espacios finales en funciones de cadena

SQL Server omite los espacios finales en los valores de cadena. Por consiguiente, al pasar espacios finales en una cadena, se pueden producir resultados imprevisibles, incluso errores.

Si tiene que tener espacios finales en la cadena, considere la posibilidad de anexar un carácter de espacio en blanco al final, de modo que SQL Server no recorte la cadena. Si no se requieren los espacios finales, se deberían recortar antes de pasarse a la canalización de la consulta.

Función RIGHT

Si se pasa un valor no `null` como primer argumento y se pasa 0 como segundo argumento a la función `RIGHT(nvarchar(max)` , 0 `)` o `RIGHT(varchar(max)` , 0 `)` , se devolverá un valor `NULL` en lugar de una cadena `empty` .

Operadores CROSS APPLY y OUTER APPLY

Los operadores CROSS APPLY y OUTER APPLY se introdujeron en SQL Server 2005. En algunos casos, la canalización de la consulta podría generar una instrucción de Transact-SQL que contenga los operadores CROSS APPLY y/o OUTER APPLY. Dado que algunos proveedores de back-end, incluidas las versiones de SQL Server anteriores a SQL Server 2005, no admiten estos operadores, tales consultas no se pueden ejecutar en estos proveedores de back-end.

A continuación se ilustran escenarios típicos que podrían conducir a la presencia de los operadores CROSS APPLY y/o OUTER APPLY en la consulta de salida:

- Una subconsulta correlacionada con paginación.
- Un `AnyElement` sobre una subconsulta correlacionada o sobre una colección generada mediante navegación.
- Consultas de LINQ que utilizan métodos de agrupación que aceptan un selector de elemento.
- Una consulta en la que se especifica explícitamente un operador CROSS APPLY u OUTER APPLY
- Una consulta que tiene una construcción DEREF sobre una construcción REF.

Operador SKIP

Si usa SQL Server 2000, el uso de SKIP con ORDER BY en columnas que no son de clave puede devolver resultados incorrectos. Se puede omitir un número superior al número especificado de filas si la columna sin clave tiene datos duplicados en ella. Esto se debe a la forma en que se traduce SKIP para SQL Server 2000. Por ejemplo, en la consulta siguiente, se pueden omitir más de cinco filas si `E.NonKeyColumn` tiene valores duplicados:

```
SELECT [E] FROM Container.EntitySet AS [E] ORDER BY [E].[NonKeyColumn] DESC SKIP 5L
```

Cuál es la versión correcta de SQL Server

La Entity Framework tiene como destino la consulta Transact-SQL basada en la versión SQL Server que se especifica en el atributo `ProviderManifestToken` del elemento Schema en el archivo de modelo de almacenamiento (. SSDL). Esta versión podría diferir de la versión de SQL Server real al que está conectado. Por ejemplo, si usa SQL Server 2005, pero el `ProviderManifestToken` atributo está establecido en 2008, la consulta Transact-SQL generada podría no ejecutarse en el servidor. Por ejemplo, una consulta que use los nuevos tipos de fecha y hora que se incluyeron en SQL Server 2008 no se ejecutará en las versiones anteriores de SQL Server. Si usa SQL Server 2005, pero el `ProviderManifestToken` atributo está establecido en 2000, la consulta Transact-SQL generada podría estar menos optimizada, o podría obtener una excepción que indica que no se admite la consulta. Para obtener más información, vea la sección Operadores CROSS APPLY y OUTER APPLY, anteriormente en este tema.

Ciertos comportamientos de las bases de datos dependen del nivel de compatibilidad establecida en la base de datos. Si el `ProviderManifestToken` atributo está establecido en 2005 y la versión de SQL Server es 2005, pero el nivel de compatibilidad de una base de datos se establece en "80" (SQL Server 2000), el valor de Transact-SQL generado será SQL Server 2005, pero es posible que no se ejecute según lo esperado debido a la configuración del nivel de compatibilidad. Por ejemplo, podría perder la información de los pedidos si un nombre de columna de la lista ORDER BY coincide con un nombre de columna en el selector.

Consultas anidadas en proyección

Las consultas anidadas en una cláusula de proyección se podrían traducir en consultas de producto cartesiano en el servidor. En algunos servidores back-end, incluido SQL Server, esto puede hacer que la tabla TempDB sea bastante grande. Esto puede hacer que disminuya el rendimiento del servidor.

El siguiente es un ejemplo de una consulta anidada en una cláusula de proyección:

```
SELECT c, (SELECT c, (SELECT c FROM AdventureWorksModel.Vendor AS c  ) As Inner2 FROM
AdventureWorksModel.JobCandidate AS c  ) As Inner1 FROM AdventureWorksModel.EmployeeDepartmentHistory AS c
```

Valores de identidad de GUID generados por el servidor

El Entity Framework admite valores de identidad de tipo GUID generados por el servidor, pero el proveedor debe admitir que se devuelva el valor de identidad generado por el servidor después de insertar una fila. A partir de SQL Server 2005, puede devolver el tipo GUID generado por el servidor en una base de datos SQL Server mediante la cláusula OUTPUT.

Consulte también

- SqlClient para Entity Framework
- Problemas conocidos y consideraciones en LINQ to Entities

Escribir un proveedor de datos de Entity Framework

23/10/2019 • 2 minutes to read • Edit Online

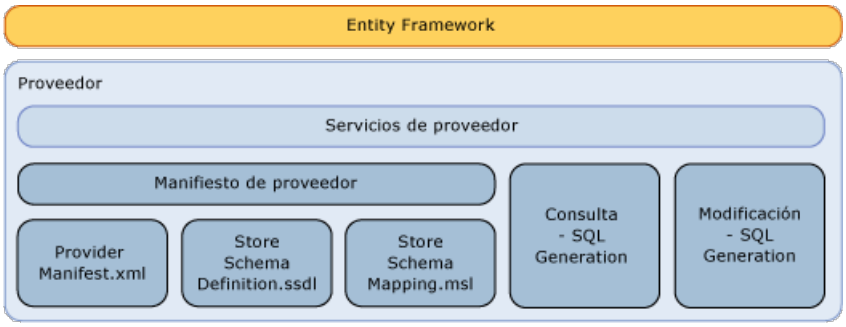
En esta sección se explica cómo escribir un proveedor de Entity Framework para admitir un origen de datos que no sea SQL Server. El Entity Framework incluye un proveedor que admite SQL Server.

Introducción al modelo de proveedor de Entity Framework

El Entity Framework es independiente de la base de datos y puede escribir un proveedor mediante el modelo de proveedor ADO.NET para conectarse a un conjunto diverso de orígenes de datos.

El proveedor de datos de Entity Framework (compilado mediante el modelo de proveedor de datos de ADO.NET) realiza las siguientes funciones:

- Asigna los tipos primitivos de Entity Data Model (EDM) a los tipos de proveedor.
- Expone funciones específicas del proveedor.
- Genera comandos específicos del proveedor para que un DbQueryCommandTree determinado admita consultas Entity Framework.
- Genera comandos de actualización específicos del proveedor para que un DbModificationCommandTree determinado admita actualizaciones a través de la Entity Framework.
- Expone archivos de asignación para la definición de esquema de almacenamiento, para admitir la generación de un modelo basado en una base de datos.
- Expone metadatos (tablas y vistas, por ejemplo) a través de un modelo conceptual.



Muestra

Vea el Entity Framework proveedor de ejemplo para obtener un ejemplo de un proveedor de Entity Framework que admite un origen de datos distinto de SQL Server.

En esta sección

Generación de SQL

Generación de SQL de modificación

Especificación del manifiesto del proveedor

Vea también

- Trabajo con proveedores de datos

Generación de SQL

23/10/2019 • 2 minutes to read • Edit Online

Al escribir un proveedor para el Entity Framework, debe traducir Entity Framework árboles de comandos en SQL que una base de datos concreta pueda comprender, como Transact-SQL para SQL Server o PL/SQL para Oracle. En esta sección, obtendrá información sobre cómo desarrollar un componente de generación de SQL (para consultas SELECT) para un proveedor de Entity Framework. Para obtener información sobre las consultas de inserción, actualización y eliminación, vea modificación de la generación de SQL.

Para entender esta sección, debe estar familiarizado con el Entity Framework y el modelo de proveedor ADO.NET. También debe comprender los árboles de comandos y DbExpression.

El rol del módulo de generación de SQL

El módulo de generación de SQL de un proveedor de Entity Framework traduce un árbol de comandos de consulta determinado en una única instrucción SELECT de SQL que tiene como destino una base de datos compatible con SQL: 1999. El SQL generado debe tener el menor número posible de consultas anidadas. El módulo de generación de SQL no debe simplificar el árbol de comandos de consulta de salida. El Entity Framework hará esto, por ejemplo, eliminando combinaciones y contraiga nodos de filtro consecutivos.

La clase DbProviderServices es el punto inicial para tener acceso al nivel de generación de SQL para convertir árboles de comandos en DbCommand.

En esta sección

Forma de los árboles de comandos

Generación de SQL a partir de árboles de comandos: procedimientos recomendados

Generación de SQL en el proveedor de ejemplo

Vea también

- Escritura de un proveedor de datos de Entity Framework

Forma de los árboles de comandos

23/10/2019 • 10 minutes to read • Edit Online

El módulo de generación de SQL es responsable de la generación de una consulta SQL back-end específica en función de una expresión determinada del árbol de comandos de consulta de entrada. En esta sección se describen las características, propiedades y estructura de los árboles de comandos de consulta.

Información general de los árboles de comandos de consulta

Un árbol de comandos de consulta es una representación del modelo de objetos de una consulta. Los árboles de comandos de consulta sirven para dos fines:

- Para expresar una consulta de entrada que se especifica en el Entity Framework.
- Expresar una consulta de salida que se proporciona a un proveedor y describe una consulta para el back-end.

Los árboles de comandos de consulta admiten una semántica más enriquecida que las consultas conformes a SQL:1999, incluso permiten trabajar con colecciones anidadas y operaciones de tipo, como por ejemplo comprobar si una entidad corresponde a un tipo determinado o filtrar conjuntos en función de un tipo.

La propiedad DBQueryCommandTree.Query es la raíz del árbol de expresiones que describe la lógica de la consulta. La propiedad DBQueryCommandTree.Parameters contiene una lista de parámetros que se utilizan en la consulta. El árbol de expresiones está compuesto por objetos DbExpression.

Un objeto DbExpression representa algún cálculo. El Entity Framework proporciona varios tipos de expresiones para crear expresiones de consulta, incluidas constantes, variables, funciones, constructores y operadores relacionales estándar como Filter y join. Cada objeto DbExpression tiene una propiedad ResultType que representa el tipo del resultado producido por esa expresión. Este tipo se expresa como TypeUsage.

Formas del árbol de comandos de consulta de salida

Los árboles de comandos de consulta de salida representan consultas relacionales (SQL) y cumplen reglas mucho más estrictas que las que se aplican a los árboles de comandos de consulta. Por lo general contienen estructuras que se traducen con facilidad a SQL.

Los árboles de comandos de entrada se expresan con respecto al modelo conceptual, que admite propiedades de navegación, asociaciones entre entidades y herencia. Los árboles de comandos de salida se expresan con respecto al modelo de almacenamiento. Los árboles de comandos de entrada permiten proyectar colecciones anidadas, pero los árboles de comandos de salida no lo permiten.

Los árboles de comandos de consulta de salida se crean utilizando un subconjunto de los objetos DbExpression disponibles e incluso algunas expresiones en este subconjunto tienen un uso restringido.

Las operaciones de tipo, como por ejemplo comprobar si una expresión determinada es de un tipo concreto o filtrar conjuntos en función de un tipo, no se encuentran en los árboles de comandos de salida.

En los árboles de comandos de salida, solo se usan expresiones que devuelven valores booleanos para las proyecciones y únicamente en predicados de expresiones que requieren un predicado, como una instrucción de filtro o una instrucción CASE.

La raíz de un árbol de comandos de consulta de salida es un objeto DbProjectExpression.

Tipos de expresión que no están presentes en el árbol de comandos de consulta de salida

Los siguientes tipos de expresión no son válidos en un árbol de comandos de consulta de salida y no necesitan ser controlados por proveedores:

- DbDerefExpression
- DbEntityRefExpression
- DbRefKeyExpression
- DbIsOfExpression
- DbOfTypeExpression
- DbRefExpression
- DbRelationshipNavigationExpression
- DbTreatExpression

Restricciones y notas de las expresiones

Muchas expresiones solo se pueden utilizar de una manera restringida en los árboles de comandos de consulta de salida:

DbFunctionExpression

Se pueden pasar los siguientes tipos de función:

- Funciones canónicas que se reconocen en el espacio de nombres Edm.
- Funciones integradas (almacén) que se reconocen en BuiltInAttribute.
- Funciones definidas por el usuario.

Las funciones canónicas (vea funciones canónicas para obtener más información) se especifican como parte de la Entity Framework y los proveedores deben proporcionar implementaciones para funciones canónicas basadas en esas especificaciones. Las funciones de almacén se basan en las especificaciones del manifiesto de proveedor correspondiente. Las funciones definidas por el usuario se basan en especificaciones de SSDL.

Además, las funciones con el atributo NiladicFunction no tienen ningún argumento y se deben traducir sin el paréntesis al final. Es decir, para *<functionname >* en lugar de *<functionname > ()* .

DbNewInstanceExpression

DbNewInstanceExpression solo se puede producir en los dos casos siguientes:

- Como la propiedad de proyección de DbProjectExpression. Cuando se utiliza de esta forma, se aplican las siguientes restricciones:
  - El tipo de resultado debe ser un tipo de fila.
  - Cada uno de sus argumentos es una expresión que genera un resultado con un tipo primitivo. Normalmente, cada argumento es una expresión escalar, como PropertyExpression en DbVariableReferenceExpression, una invocación de función o un cálculo aritmético de DbPropertyExpression en DbVariableReferenceExpression o una invocación de función. Sin embargo, una expresión que representa una subconsulta escalar también se puede producir en la lista de argumentos para DbNewInstanceExpression. Una expresión que representa una subconsulta escalar es un árbol de expresión que representa una subconsulta que devuelve exactamente una fila y una columna de un tipo primitivo con una raíz del objeto DbElementExpression
- Con un tipo de valor devuelto de colección, en cuyo caso define una nueva colección de las expresiones proporcionadas como argumentos.

DbVariableReferenceExpression

DbVariableReferenceExpression tiene que ser un elemento secundario del nodo DbPropertyExpression.

DbGroupByExpression

La propiedad Aggregates de DbGroupByExpression solo puede tener elementos de tipo DbFunctionAggregate. No hay ningún otro tipo de agregado.

DbLimitExpression

La propiedad Limite solo puede ser DbConstantExpression o DbParameterReferenceExpression. Además, el valor de la propiedad WithTies es siempre false a partir de la versión 3.5 de .NET Framework.

DbScanExpression

Cuando se usa en árboles de comandos de salida, DbScanExpression representa eficazmente un recorrido en una tabla, una vista o una consulta de almacén, representada por EntitySetBase:: target.

Si la propiedad de metadatos "definiendo consulta" del destino no es null, representa una consulta, el texto de la consulta para el que se proporciona en esa propiedad de metadatos en el idioma específico del proveedor (o dialecto) tal y como se especifica en la definición del esquema de almacenamiento.

De lo contrario, el destino representa una tabla o una vista. El prefijo de esquema está en la propiedad de metadatos "Schema", si no es null; de lo contrario, es el nombre del contenedor de entidades. El nombre de la tabla o vista es la propiedad de metadatos "Table", si no es null; de lo contrario, es la propiedad Name de la base del conjunto de entidades.

Todas estas propiedades proceden de la definición del elemento EntitySet correspondiente en el archivo de definición de esquemas de almacenamiento (SSDL).

Usar tipos primitivos

Cuando se hace referencia a los tipos primitivos en los árboles de comandos de salida, normalmente se hace referencia a ellos en los tipos primitivos del modelo conceptual. Sin embargo, para ciertas expresiones, los proveedores necesitan el tipo primitivo de almacén correspondiente. Entre los ejemplos de estas expresiones se encuentran DbCastExpression y posiblemente DbNullExpression, si el proveedor necesita convertir el valor NULL al tipo correspondiente. En estos casos, los proveedores deben realizar la asignación al tipo de proveedor en función de la clase del tipo primitivo y sus facetas.

Vea también

- Generación de SQL

Generar SQL a partir de árboles de comandos: procedimientos recomendados

29/10/2019 • 10 minutes to read • Edit Online

Los árboles de comandos de consulta de salida crean modelos muy similares a consultas expresables en SQL. Sin embargo, hay ciertos retos comunes para los programadores de proveedores a la hora de generar SQL a partir de un árbol de comandos de salida. En este tema se describen estos retos. En el tema siguiente, el proveedor de ejemplo muestra cómo actuar ante estos retos.

Agrupar nodos DbExpression en una instrucción SELECT de SQL

Una instrucción SQL típica tiene una estructura anidada con la siguiente forma:

```
SELECT …
FROM …
WHERE …
GROUP BY …
ORDER BY …
```

Una o más cláusulas pueden estar vacías. Una instrucción SELECT anidada puede producirse en cualquiera de las líneas.

Una posible traducción de un árbol de comandos de consulta en una instrucción SELECT de SQL generaría una subconsulta para cada operador relacional. Sin embargo, esto conduciría a subconsultas anidadas innecesarias que serían difíciles de leer. En algunos almacenes de datos, esta consulta puede tener un bajo rendimiento.

Como ejemplo, observe el siguiente árbol de comandos de consulta:

```
Project (
a.x,
   a = Filter(
      b.y = 5,
      b = Scan("TableA")
   )
)
```

Una traducción ineficaz generaría:

```
SELECT a.x
FROM (   SELECT *
         FROM TableA as b
         WHERE b.y = 5) as a
```

Observe que cada nodo de expresión relacional se convierte en una nueva instrucción SELECT de SQL.

Por tanto, es importante agregar tantos nodos de expresión como sea posible en una única instrucción SELECT de SQL conservando la exactitud.

El resultado de esta agregación en el ejemplo presentado anteriormente sería:

```
SELECT b.x
FROM TableA as b
WHERE b.y = 5
```

Reducir combinaciones en una instrucción SELECT de SQL

Un caso de agregación de varios nodos en una única instrucción SELECT de SQL consiste en agregar varias expresiones de combinación en una única instrucción SELECT de SQL. DbJoinExpression representa una combinación única entre dos entradas. Sin embargo, como parte de una instrucción SELECT de SQL única, se puede especificar más de una combinación. En este caso, las combinaciones se realizan en el orden especificado.

Las combinaciones del lateral izquierdo (combinaciones que aparecen como elemento secundario izquierdo de otra combinación) se pueden reducir más fácilmente a una instrucción SELECT de SQL única. Por ejemplo, observe el siguiente árbol de comandos de consulta:

```
InnerJoin(
   a = LeftOuterJoin(
   b = Extent("TableA")
   c = Extent("TableB")
   ON b.y = c.x ),
   d = Extent("TableC")
   ON a.b.y = d.z
)
```

Esto se puede traducir correctamente en:

```
SELECT *
FROM TableA as b
LEFT OUTER JOIN TableB as c ON b.y = c.x
INNER JOIN TableC as d ON b.y = d.z
```

Sin embargo, las combinaciones no situadas en el lateral izquierdo no se pueden reducir fácilmente y no debe intentar llevar a cabo esta acción. Por ejemplo, las combinaciones del siguiente árbol de comandos de consulta:

```
InnerJoin(
   a = Extent("TableA")
   b = LeftOuterJoin(
   c = Extent("TableB")
   d = Extent("TableC")
   ON c.y = d.x),
   ON a.z = b.c.y
)
```

Se traducirían a una instrucción SELECT de SQL con una subconsulta.

```
SELECT *
FROM TableA as a
INNER JOIN (SELECT *
   FROM TableB as c
   LEFT OUTER JOIN TableC as d
   ON c.y = d.x) as b
ON b.y = d.z
```

Redirección de alias de entrada

Para explicar la redirección de alias de entrada, considere la estructura de las expresiones relacionales, como DbFilterExpression, DbProjectExpression, DbCrossJoinExpression, DbJoinExpression, DbSortExpression, DbGroupByExpression, DbApplyExpression y DbSkipExpression.

Cada uno de estos tipos tiene una o más propiedades Input que describen una colección de entradas, y se usa una variable de enlace correspondiente a cada entrada para representar cada elemento de dicha entrada durante un recorrido de la colección. La variable de enlace se usa al hacer referencia al elemento de entrada, por ejemplo en la propiedad Predicado de DbFilterExpression o en la propiedad Projection de DbProjectExpression.

Al agregar más nodos de expresión relacional en una única instrucción SELECT de SQL y evaluar una expresión que forma parte de una expresión relacional (por ejemplo parte de la propiedad Projection de DbProjectExpression), es posible que la variable de enlace que utiliza no sea igual que el alias de la entrada, ya que se redirigen varios enlaces de expresión a una única extensión. Este problema se denomina cambio de nombre de alias.

Considere el primer ejemplo de este tema. Si se realiza una traducción básica y se traduce Projection a.x (DbPropertyExpression (a, x)), es correcto traducirlo como `a.x`, ya que hemos definido la entrada con el alias "a" para coincidir con la variable de enlace. Sin embargo, al agregar ambos nodos en una instrucción SELECT de SQL única, debe traducir el mismo elemento DbPropertyExpression a `b.x`, ya que la entrada se ha definido con el alias "b".

Eliminación de la información de estructura jerárquica de los alias de combinación

A diferencia de cualquier otra expresión relacional en un árbol de comandos de salida, DbJoinExpression genera un tipo de resultado que es una fila compuesta de dos columnas, cada una de las cuales corresponde a una de las entradas. Cuando se crea una DbPropertyExpression para tener acceso a una propiedad escalar que se origina a partir de una combinación, se trata de otro DbPropertyExpression.

En los ejemplos se incluye "a.b.y" en el ejemplo 2 y "b.c.y" en el ejemplo 3. Sin embargo, en las instrucciones SQL correspondientes se hace referencia a ellos como "b.y". Este nuevo uso de alias se denomina reducción de alias de combinación.

Cambio de nombre de una columna y de un alias de extensión

Si una consulta SELECT de SQL que tiene una combinación se debe completar con una proyección, al enumerar todas las columnas que participan de las entradas, se puede producir un conflicto de nombres, ya que varias entradas pueden tener el mismo nombre de columna. Utilice un nombre diferente en la columna para evitar el conflicto.

También, al reducir las combinaciones, las tablas (o subconsultas) que participan pueden usar alias en conflicto, en cuyo caso será necesario cambiar su nombre.

Evitar SELECT *

No utilice `SELECT *` para seleccionar en las tablas bases. El modelo de almacenamiento de una aplicación Entity Framework puede incluir solo un subconjunto de las columnas que se encuentran en la tabla de base de datos. En este caso, `SELECT *` puede generar un resultado incorrecto. En su lugar, debe especificar todas las columnas que participan utilizando los nombres de columna del tipo de resultado de las expresiones que participan.

Reutilizar expresiones

Las expresiones se pueden reutilizar en el árbol de comandos de consulta pasado por el Entity Framework. No suponga que cada expresión aparece una sola vez en el árbol de comandos de consulta.

Asignar tipos primitivos

Al asignar tipos conceptuales (EDM) a los tipos de proveedor, debe asignar al tipo más ancho (Int32) para que quepan todos los valores posibles. Además, evite la asignación a tipos que no se pueden usar para muchas operaciones, como los tipos de BLOB (por ejemplo, `ntext` en SQL Server).

Vea también

- Generación de SQL

Generación de SQL en el proveedor de ejemplo

23/10/2019 • 2 minutes to read • Edit Online

En el Entity Framework proveedor de ejemplo se muestran los nuevos componentes de los proveedores de datos de ADO.net que admiten el Entity Framework. Funciona con una base de datos SQL Server 2005 y se implementa como un contenedor del proveedor de datos ADO.NET 2.0 System.Data.SqlClient.

El módulo Generación de SQL del proveedor de ejemplo (ubicado en la carpeta Generación de SQL, salvo el archivo DmlSqlGenerator.cs) toma una entrada DbQueryCommandTree y genera una instrucción SQL SELECT única.

En esta sección

Esta sección contiene los siguientes temas:

Arquitectura y diseño

Tutorial: Generación de SQL

Vea también

- Generación de SQL

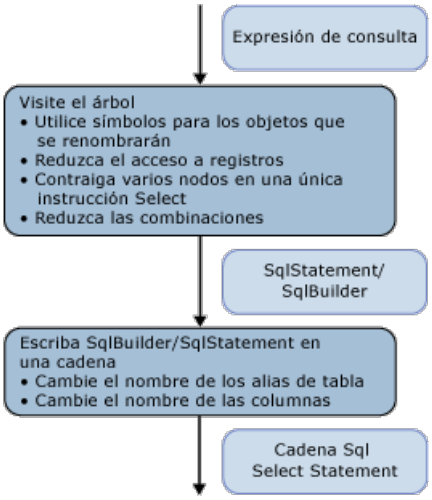
Arquitectura y diseño

29/10/2019 • 34 minutes to read • Edit Online

El módulo de generación de SQL del proveedor de ejemplo se implementa como visitante en el árbol de expresión que representa el árbol de comandos. La generación se realiza en un paso único al árbol de expresión.

Los nodos del árbol se procesan de abajo arriba. Primero se genera una estructura intermedia, SqlSelectStatement o SqlBuilder, que implementan ISqlFragment. A continuación, la instrucción SQL de la cadena se genera a partir de esa estructura. Hay dos motivos para la estructura intermedia:

- Lógicamente, una instrucción SQL SELECT se rellena de manera desordenada. Los nodos que participan en la cláusula FROM se visitan antes que los nodos que participan en las cláusulas WHERE, GROUP BY y ORDER BY.
- Para cambiar el nombre de los alias, debe identificar todos los alias que se hayan utilizado para evitar colisiones durante el cambio de nombre. Para aplazar las opciones de cambio de nombre en SqlBuilder, utilice objetos Symbol para representar las columnas candidatas al cambio de nombre.



En la primera fase, cuando se visita el árbol de expresión, las expresiones se agrupan en instrucciones SqlSelectStatement y se elimina la información de estructura jerárquica de las combinaciones y de los alias de combinación. Durante esta fase, los objetos Symbol representan columnas o alias de entrada a los que se puede cambiar el nombre.

En la segunda fase, cuando se genera la cadena real, se cambia el nombre de los alias.

Estructuras de datos

En esta sección se describen los tipos utilizados en el proveedor de ejemplo que se utilizan para generar una instrucción SQL.

ISqlFragment

Esta sección abarca las clases que implementan la interfaz ISqlFragment, que sirve para dos fines:

- Un tipo de valor devuelto común para todos los métodos de visitante.
- Proporciona un método para escribir la cadena de SQL final.

```
internal interface ISqlFragment {
   void WriteSql(SqlWriter writer, SqlGenerator sqlGenerator);
}
```

SqlBuilder

SqlBuilder es un dispositivo de recopilación para la cadena de SQL final, similar a StringBuilder. Está compuesto de las cadenas que constituyen el SQL final, junto con objetos ISqlFragment que se pueden convertir en cadenas.

```
internal sealed class SqlBuilder : ISqlFragment {
   public void Append(object s)
   public void AppendLine()
   public bool IsEmpty
}
```

SqlSelectStatement

Instrucción sqlselectstatement representa una instrucción SQL SELECT canónica de la forma "SELECT... De.. DÓNDE... AGRUPAR POR... ORDER BY ".

Cada una de las cláusulas de SQL está representada por un objeto StringBuilder. Además, realiza un seguimiento de si se ha especificado Distinct y si la instrucción se encuentra en el nivel más alto. Si la instrucción no se encuentra en el nivel más alto, la cláusula ORDER BY se omite, a menos que la instrucción también tenga una cláusula TOP.

FromExtents contiene la lista de entradas de la instrucción SELECT. Normalmente, consta de un único elemento. Las instrucciones SELECT de las combinaciones pueden tener temporalmente más de un elemento.

Si un nodo Join crea la instrucción SELECT, SqlSelectStatement mantiene una lista de todas las extensiones cuya información de estructura jerárquica se ha eliminado en la combinación en AllJoinExtents. OuterExtents representa las referencias externas de la instrucción SqlSelectStatement y se utiliza para cambiar el nombre de los alias de entrada.

```
internal sealed class SqlSelectStatement : ISqlFragment {
   internal bool IsDistinct { get, set };
   internal bool IsTopMost

   internal List<Symbol> AllJoinExtents { get, set };
   internal List<Symbol> FromExtents { get};
   internal Dictionary<Symbol, bool> OuterExtents { get};

   internal TopClause Top { get, set };

   internal SqlBuilder Select {get};
   internal SqlBuilder From
   internal SqlBuilder Where
   internal SqlBuilder GroupBy
   public SqlBuilder OrderBy
}
```

TopClause

TopClause representa la expresión TOP en una instrucción SqlSelectStatement. La propiedad TopCount indica cuántas filas TOP deben seleccionarse. Cuando WithTies es true, la cláusula se compiló a partir de un DbLimitExpression.

```
class TopClause : ISqlFragment {
   internal bool WithTies {get}
   internal ISqlFragment TopCount {get}
   internal TopClause(ISqlFragment topCount, bool withTies)
   internal TopClause(int topCount, bool withTies)
}
```

Símbolos

Las clases relacionadas con Symbol y la tabla de símbolos cambian el nombre a los alias de entrada, eliminan información de estructura jerárquica de los alias de combinación y cambian el nombre a los alias de columna.

La clase Symbol representa una extensión, una instrucción SELECT anidada o una columna. Se utiliza en lugar de un alias real para permitir el cambio de nombre una vez utilizada y presenta también información adicional del artefacto que representa (como el tipo).

```
class Symbol : ISqlFragment {
   internal Dictionary<string, Symbol> Columns {get}
   internal bool NeedsRenaming {get, set}
   internal bool IsUnnest {get, set}   //not used

   public string Name{get}
   public string NewName {get,set}
   internal TypeUsage Type {get, set}

   public Symbol(string name, TypeUsage type)
}
```

El nombre almacena el alias original de la extensión representada, la instrucción SELECT anidada o una columna.

NewName almacena el alias que se utilizará en la instrucción SELECT de SQL. Se establece originalmente en Name y solo se cambia el nombre si resulta necesario al generar la consulta de la cadena final.

El tipo solamente es útil para los símbolos que representan extensiones e instrucciones SELECT anidadas.

SymbolPair

La clase SymbolPair se encarga de la eliminación de la información de estructura jerárquica de los registros.

Tomemos como ejemplo una expresión de propiedad D(v, "j3.j2.j1.a.x"), donde v es una VarRef, j1, j2, j3 son las combinaciones, a es una extensión y x es una columna.

Esto tiene que ser traducido posteriormente a {j'}.{x'}. El campo de origen representa la instrucción SqlStatement extrema, que representa una expresión de combinación (por ejemplo, j2); es siempre un símbolo Join. El campo de columna se mueve de un símbolo de combinación al siguiente hasta que se detiene en un símbolo que no es de combinación. Es lo que se devuelve cuando se visita una expresión DbPropertyExpression, pero nunca se agrega a un SqlBuilder.

```
class SymbolPair : ISqlFragment {
   public Symbol Source;
   public Symbol Column;
   public SymbolPair(Symbol source, Symbol column)
}
```

JoinSymbol

Un símbolo Join es un objeto Symbol que representa una instrucción SELECT anidada con una combinación o una entrada de combinación.

```
internal sealed class JoinSymbol : Symbol {
   internal List<Symbol> ColumnList {get, set}
   internal List<Symbol> ExtentList {get}
   internal List<Symbol> FlattenedExtentList {get, set}
   internal Dictionary<string, Symbol> NameToExtent {get}
   internal bool IsNestedJoin {get, set}

   public JoinSymbol(string name, TypeUsage type, List<Symbol> extents)
}
```

ColumnList representa la lista de columnas de la cláusula SELECT si este símbolo representa una instrucción SELECT de SQL. ExtentList es la lista de extensiones de la cláusula SELECT. Si la combinación tiene varias extensiones sin información de estructura jerárquica en el nivel superior, FlattenedExtentList realiza un seguimiento de las extensiones para asegurarse de que el nombre de los alias de extensión se cambia correctamente.

NameToExtent tiene todas las extensiones de ExtentList como un diccionario. IsNestedJoin se utiliza para determinar si JoinSymbol es un símbolo de combinación normal o un símbolo con una instrucción SqlSelectStatement correspondiente.

Todas las listas se establecen una vez exactamente y después se utilizan para realizar búsquedas o para enumeraciones.

SymbolTable

SymbolTable se utiliza para resolver nombres de variable como símbolos. SymbolTable se implementa como una pila con una nueva entrada para cada ámbito. Las búsquedas examinan la pila desde la parte superior a la parte inferior hasta que se encuentra una entrada.

```
internal sealed class SymbolTable {
   internal void EnterScope()
   internal void ExitScope()
   internal void Add(string name, Symbol value)
   internal Symbol Lookup(string name)
}
```

Hay solo una tabla SymbolTable por instancia del módulo de generación de Sql. Se entra y se sale de los ámbitos para cada nodo relacional. Todos los símbolos de ámbitos anteriores están visibles para los ámbitos posteriores, a menos que estén ocultos por otros símbolos con el mismo nombre.

Estado global para el visitante

Para ayudar en el cambio de nombre de alias y columnas, mantenga una lista de todos los nombres de columna (AllColumnNames) y alias de extensión (AllExtentNames) utilizados en el primer paso al árbol de consultas. La tabla de símbolos resuelve los nombres de variable como símbolos. IsVarRefSingle solo se utiliza a efectos de comprobación, no es estrictamente necesario.

Las dos pilas utilizadas mediante CurrentSelectStatement e IsParentAJoin se utilizan para pasar los "parámetros" de los nodos primarios a los secundarios, puesto que el patrón del visitante no permite el paso de parámetros.

```
internal Dictionary<string, int> AllExtentNames {get}
internal Dictionary<string, int> AllColumnNames {get}
SymbolTable symbolTable = new SymbolTable();
bool isVarRefSingle = false;

Stack<SqlSelectStatement> selectStatementStack;
private SqlSelectStatement CurrentSelectStatement{get}

Stack<bool> isParentAJoinStack;
private bool IsParentAJoin{get}
```

Escenarios comunes

En esta sección se explican los escenarios de proveedor más habituales.

Agrupar nodos de expresión en instrucciones SQL

Cuando se encuentra el primer nodo relacional (normalmente una extensión de DbScanExpression) al visitar el árbol de abajo arriba, se crea una instrucción SqlSelectStatement. Para generar una instrucción SELECT de SQL con el menor número de consultas anidadas posible, agregue tantos nodos primarios como sea posible en esa instrucción SqlSelectStatement.

La decisión sobre si un determinado nodo (relacional) se puede agregar a la instrucción SqlSelectStatement actual (la que se devolvió al visitar la entrada) o si es necesario iniciar una nueva instrucción es calculada por el método IsCompatible y depende del contenido de la instrucción SqlSelectStatement, que depende, a su vez, de qué nodos estaban situados por debajo del nodo especificado.

Normalmente, si las cláusulas de instrucciones SQL se evalúan después de las cláusulas en las que los nodos que se están considerando para la combinación no están vacíos, el nodo no se puede agregar a la instrucción actual. Por ejemplo, si el nodo siguiente es un objeto Filter, solo se podrá incorporar a la instrucción SqlSelectStatement actual si se cumple lo siguiente:

- La lista SELECT está vacía. Si no lo está, un nodo anterior al filtro generó la lista de selección y puede que el predicado haga referencia a las columnas generadas por esa lista SELECT.
- GROUPBY está vacío. Si no lo está, agregar el filtro significaría que el filtrado se produciría antes que la agrupación, lo cual no es correcto.
- La cláusula TOP está vacía. Si no lo está, agregar el filtro significaría que el filtrado se produciría antes que la ejecución de la cláusula TOP, lo cual no es correcto.

Esto no se aplica a los nodos no relacionales, como DbConstantExpression, o a expresiones aritméticas, porque se incluyen siempre como parte de una instrucción SqlSelectStatement existente.

Asimismo, al encontrar la raíz del árbol de combinaciones (un nodo de combinación que no tiene un elemento primario de combinación), se inicia una nueva instrucción SqlSelectStatement. Todos los elementos secundarios de combinación del lateral izquierdo se agregan a esa instrucción SqlSelectStatement.

Siempre que se inicia una instrucción SqlSelectStatement nueva, y la actual se agrega a la entrada, es posible que la instrucción SqlSelectStatement actual necesite completarse agregando columnas de proyección (una cláusula SELECT) si no existe una. Esto se realiza con el método AddDefaultColumns, que examina la propiedad FromExtents de la instrucción SqlSelectStatement y agrega todas las columnas que la lista de extensiones representada por FromExtents introduce en el ámbito de la lista de columnas proyectadas. El motivo de esto es porque en ese punto no se sabe a qué columnas hacen referencia los otros nodos. Esto se puede optimizar para proyectar solo las columnas que se puedan utilizar después.

Eliminación de la información de estructura jerárquica de las combinaciones

La propiedad IsParentAJoin ayuda a determinar si se puede quitar la información de estructura jerárquica de una determinada combinación. En particular, la propiedad IsParentAJoin solo devuelve `true` para el elemento secundario izquierdo de una combinación y para cada DbScanExpression que sea una entrada inmediata de una combinación, en cuyo caso ese nodo secundario vuelve a utilizar la misma instrucción SqlSelectStatement que el elemento primario utilizaría después. Para obtener más información acerca de las expresiones, vea "Expresiones de combinación".

Redirección de alias de entrada

La redirección de los alias de entrada se logra con la tabla de símbolos.

Para explicar el redireccionamiento de los alias de entrada, consulte el primer ejemplo de generación de SQL a partir de árboles de comandos: procedimientos recomendados. En él, "a" necesita redirigirse a "b" en la proyección.

Cuando se crea un objeto SqlSelectStatement, la extensión que representa la entrada al nodo se coloca en la propiedad From del objeto SqlSelectStatement. Se crea un símbolo (<symbol_b >) basándose en el nombre del enlace de entrada ("b") para representar esa extensión y "AS" + <symbol_b > se anexa a la cláusula FROM. El símbolo también se agrega a la propiedad FromExtents.

El símbolo también se agrega a la tabla de símbolos para vincularle el nombre del enlace de entrada ("b", <symbol_b >).

Si un nodo posterior vuelve a utilizar esa instrucción SqlSelectStatement, agrega una entrada a la tabla de símbolos para vincular su nombre de enlace de entrada a ese símbolo. En nuestro ejemplo, el DbProjectExpression con el nombre de enlace de entrada de "a" reutilizaría instrucción sqlselectstatement y agregaría ("a" < symbol_b >) a la tabla.

Cuando las expresiones hacen referencia al nombre de enlace de entrada del nodo que está reutilizando la instrucción SqlSelectStatement, esa referencia se resuelve usando la tabla de símbolos como el símbolo redirigido correcto. Cuando "a" de "a. x" se resuelve al visitar el DbVariableReferenceExpression que representa "a", se resolverá en el símbolo <symbol_b >.

Eliminación de la información de estructura jerárquica de los alias de combinación

La eliminación de la información de estructura jerárquica de los alias de combinación se logra al visitar una expresión DbPropertyExpression, como se describe en la sección titulada DbPropertyExpression.

Cambio de nombre de una columna y de un alias de extensión

El problema del cambio de nombre de una columna y de un alias de extensión se resuelve utilizando símbolos que solo se sustituyen por alias en la segunda fase de la generación, como se describe en la sección titulada Segunda fase de la generación de SQL: Generar el comando String.

Primera fase de la generación de SQL: Visitar el árbol de expresión

En esta sección se describe la primera fase de generación de SQL, cuando se visita la expresión que representa la consulta y se genera una estructura intermedia, SqlSelectStatement o SqlBuilder.

En esta sección se describen los principios de las visitas a las diferentes categorías de nodo de expresión y se ofrecen detalles sobre las visitas a determinados tipos de expresión.

Nodos relacionales (no de combinación)

Los siguientes tipos de expresión admiten nodos que no son de combinación:

- DbDistinctExpression
- DbFilterExpression
- DbGroupByExpression
- DbLimitExpression
- DbProjectExpression
- DbSkipExpression
- DbSortExpression

La visita a estos nodos sigue este patrón:

1. Visite la entrada relacional y obtenga la instrucción SqlSelectStatement resultante. La entrada a un nodo relacional puede ser:
    - Un nodo relacional, incluida una extensión (una expresión DbScanExpression, por ejemplo). Al visitar este tipo de nodo, se devuelve una instrucción SqlSelectStatement.

- Una expresión de operación set (UNION ALL, por ejemplo). El resultado tiene que incluirse entre corchetes y colocarse en la cláusula FROM de una nueva instrucción SqlSelectStatement.

2. Compruebe si el nodo actual se puede agregar a la instrucción SqlSelectStatement generada por la entrada. Esto se describe en la sección titulada Agrupar expresiones en instrucciones SQL. En caso contrario,
   - Extraiga el objeto SqlSelectStatement actual.
   - Cree un nuevo objeto SqlSelectStatement y agregue el objeto SqlSelectStatement extraído como la cláusula FROM del nuevo objeto SqlSelectStatement.
   - Coloque el nuevo objeto en la parte superior de la pila.
3. Redirija el enlace de expresión de entrada al símbolo correcto de la entrada. Esta información se conserva en el objeto SqlSelectStatement.
4. Agregue un nuevo ámbito SymbolTable.
5. Visite la parte que no es de entrada de la expresión (por ejemplo, Projection y Predicado).
6. Extraiga todos los objetos agregados a las pilas globales.

DbSkipExpression no tiene un equivalente directo en SQL. Lógicamente, se traduce como:

```
SELECT Y.x1, Y.x2, ..., Y.xn
FROM (
   SELECT X.x1, X.x2, ..., X.xn, row_number() OVER (ORDER BY sk1, sk2, ...) AS [row_number]
   FROM input as X
   ) as Y
WHERE Y.[row_number] > count
ORDER BY sk1, sk2, ...
```

Expresiones de combinación

Las siguientes expresiones están consideradas como de combinación y se procesan de manera ordinaria utilizando el método VisitJoinExpression:

- DbApplyExpression
- DbJoinExpression
- DbCrossJoinExpression

A continuación se enumeran los pasos que deben seguirse en la visita:

Primero, antes de visitar los elementos secundarios, es necesario llamar a IsParentAJoin para comprobar si el nodo de combinación es un elemento secundario de una combinación situada en un lateral izquierdo. Si devuelve false, se inicia una nueva instrucción SqlSelectStatement. En ese sentido, las combinaciones se visitan de forma diferente al resto de los nodos, ya que el elemento primario (el nodo de combinación) crea la instrucción SqlSelectStatement para que la usen posiblemente los elementos secundarios.

En segundo lugar, procese las entradas de una en una. Para cada entrada:

1. Visite la entrada.
2. Posprocese el resultado de la visita a la entrada llamando a ProcessJoinInputResult, que se encarga de mantener la tabla de símbolos después de visitar un elemento secundario de una expresión de combinación y finalizar posiblemente la instrucción SqlSelectStatement generada por el elemento secundario. El resultado del elemento secundario puede ser:
   - Un objeto SqlSelectStatement distinto del objeto al que se agregará el elemento principal. En este

caso, puede que resulte necesario completarlo agregando columnas predeterminadas. Si la entrada fue un objeto Join, tendrá que crear un nuevo símbolo de combinación. De lo contrario, cree un símbolo normal.

- Una extensión (una expresión DbScanExpression, por ejemplo), en cuyo caso simplemente se agrega a la lista de entradas de la instrucción SqlSelectStatement del elemento primario.
- Un objeto que no es SqlSelectStatement, en cuyo caso se coloca entre corchetes.
- El mismo objeto SqlSelectStatement al que se agrega el elemento primario. En este caso, los símbolos de la lista FromExtents deben sustituirse por un nuevo objeto JoinSymbol único que los represente a todos.
- En estos tres primeros casos, se llama a AddFromSymbol para agregar la cláusula AS y actualizar la tabla de símbolos.

En tercer lugar, se visita la condición de combinación (si existe).

Operaciones Set

El método VisitSetOpExpression procesa las operaciones set DbUnionAllExpression, DbExceptExpression y DbIntersectExpression. Crea un objeto SqlBuilder del tipo

```
<leftSqlSelectStatement> <setOp> <rightSqlSelectStatement>
```

Donde <leftSqlSelectStatement > y <> rightSqlSelectStatement se obtienen instrucciones sqlselectstatement mediante la visita de cada una de las entradas y <setOp > es la operación correspondiente (UNION ALL, por ejemplo).

DbScanExpression

Si se visita en un contexto de combinación (como una entrada a una combinación que representa un elemento secundario izquierdo de otra combinación), la expresión DbScanExpression devuelve un objeto SqlBuilder con el SQL de destino del correspondiente destino, que puede ser una consulta de definición, una tabla o una vista. De lo contrario, se crea una instrucción SqlSelectStatement con el campo FROM establecido para corresponderse con el destino correspondiente.

DbVariableReferenceExpression

La visita de una expresión DbVariableReferenceExpression devuelve el objeto Symbol correspondiente a esa expresión de referencia de variable en función de una búsqueda en la tabla de símbolos.

DbPropertyExpression

La eliminación de la información de estructura jerárquica de los alias de combinación se identifica y procesa al visitar una expresión DbPropertyExpression.

Primero se visita la propiedad Instance, lo que da como resultado un objeto Symbol, JoinSymbol o SymbolPair. A continuación se muestra cómo se tratan estos tres casos:

- Si se devuelve un objeto JoinSymbol, su propiedad NameToExtent contiene un símbolo de la propiedad necesaria. Si el símbolo de la combinación representa una combinación anidada, se devuelve un nuevo par Symbol con el símbolo de la combinación para realizar un seguimiento del símbolo que podría utilizarse como alias de la instancia y el símbolo que representa la propiedad real para una resolución posterior.
- Si se devuelve un objeto SymbolPair y la parte Column es un símbolo de combinación, se devuelve de nuevo un símbolo de combinación, pero ahora la propiedad de columna se actualiza para señalar a la propiedad representada por la expresión de propiedad actual. De lo contrario, se devuelve un objeto SqlBuilder con el origen de SymbolPair como alias y el símbolo de la propiedad actual como columna.

- Si se devuelve un objeto Symbol, el método Visit devuelve un método SqlBuilder con esa instancia como alias y el nombre de propiedad como nombre de columna.

DbNewInstanceExpression

Cuando se utiliza como la propiedad Projection de DbProjectExpression, DbNewInstanceExpression genera una lista separada por comas de los argumentos para representar las columnas proyectadas.

Cuando DbNewInstanceExpression tiene un tipo de valor devuelto de colección, y define una nueva colección de las expresiones proporcionadas como argumentos, los tres casos siguientes se tratan por separado:

- Si DbNewInstanceExpression tiene DbElementExpression como único argumento, se traduce como:

```
NewInstance(Element(X)) =>  SELECT TOP 1 …FROM X
```

Si DbNewInstanceExpression no tiene ningún argumento (representa una tabla vacía), DbNewInstanceExpression se traduce como:

```
SELECT CAST(NULL AS <primitiveType>) as X
FROM (SELECT 1) AS Y WHERE 1=0
```

De lo contrario, DbNewInstanceExpression genera una escala union-all de los argumentos:

```
SELECT <visit-result-arg1> as X
UNION ALL SELECT <visit-result-arg2> as X
UNION ALL …
UNION ALL SELECT <visit-result-argN> as X
```

DbFunctionExpression

Las funciones canónicas e integradas se procesan de la misma manera: si necesitan un control especial (TRIM(cadena) como LTRIM(RTRIM(cadena), por ejemplo), se invoca al controlador adecuado. De lo contrario, se traducen como FunctionName(arg1, arg2, ..., argn).

Se utilizan diccionarios para realizar un seguimiento de qué funciones necesitan un control especial y sus controladores adecuados.

Las funciones definidas por el usuario se traducen a NamespaceName. FunctionName (arg1, arg2,..., argn).

DbElementExpression

El método que visita DbElementExpression solamente se invoca para visitar una expresión DbElementExpression cuando se usa para representar una subconsulta escalar. Por consiguiente, DbElementExpression se traduce en una instrucción SqlSelectStatement completa y agrega corchetes alrededor de ella.

DbQuantifierExpression

Dependiendo del tipo de expresión (any o All), DbQuantifierExpression se traduce como:

```
Any(input, x) => Exists(Filter(input,x))
All(input, x) => Not Exists(Filter(input, not(x))
```

DbNotExpression

En algunos casos, es posible contraer la traducción de DbNotExpression con su expresión de entrada. Por ejemplo:

```
Not(IsNull(a)) =>  "a IS NOT NULL"
Not(All(input, x) => Not (Not Exists(Filter(input, not(x))) => Exists(Filter(input, not(x))
```

El motivo por el que se contrae por segunda vez es porque el proveedor introdujo ineficacias al traducir una expresión DbQuantifierExpression de tipo All. Por consiguiente, Entity Framework no podría haber realizado la simplificación.

DbIsEmptyExpression

DbIsEmptyExpression se traduce como:

```
IsEmpty(input) = Not Exists(input)
```

Segunda fase de la generación de SQL: Generar el comando String

Al generar un comando SQL string, el objeto SqlSelectStatement genera los alias reales de los símbolos, lo que resuelve el problema del cambio de nombre de la columna y del alias de extensión.

El cambio de nombre del alias de extensión se produce al escribir el objeto SqlSelectStatement en una cadena. Primero, cree una lista de todos los alias utilizados por las extensiones exteriores. El nombre de cada símbolo en la propiedad FromExtents (o en AllJoinExtents si no es null) se cambia si entra en colisión con alguna de las extensiones exteriores. Si se necesita un cambio de nombre, no estará en conflicto con ninguna de las extensiones recopiladas en AllExtentNames.

El cambio de nombre de una columna se produce cuando se escribe un objeto Symbol en una cadena. AddDefaultColumns en la primera fase ha determinado si debe cambiarse el nombre de un determinado símbolo de columna. En la segunda fase, el cambio de nombre solamente se produce tras asegurarse de que el nombre generado no estará en conflicto con ninguno de los nombres utilizados en AllColumnNames

Para generar nombres únicos tanto para los alias de extensión como para las columnas, use <existing_name > _n, donde n es el alias más pequeño que todavía no se ha usado. La lista global de todos los alias aumenta la necesidad de realizar cambios de nombre en cascada.

Vea también

- Generación de SQL en el proveedor de ejemplo

Tutorial: Generar SQL

29/10/2019 • 17 minutes to read • Edit Online

En este tema se muestra cómo se produce la generación de SQL en el proveedor de ejemplo. La siguiente consulta de Entity SQL utiliza el modelo incluido con el proveedor de ejemplo:

```
SELECT  j1.ProductId, j1.ProductName, j1.CategoryName, j2.ShipCountry, j2.ProductId
FROM (  SELECT P.ProductName, P.ProductId, P.Category.CategoryName
        FROM NorthwindEntities.Products AS P) as j1
INNER JOIN (SELECT OD.ProductId, OD.Order.ShipCountry as ShipCountry
            FROM NorthwindEntities.OrderDetails AS OD) as j2
            ON j1.ProductId == j2.ProductId
```

La consulta genera el siguiente árbol de comandos de salida que se pasa al proveedor:

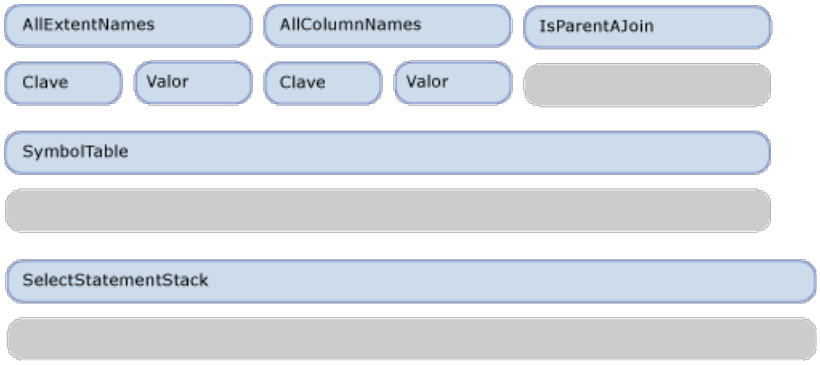
```
DbQueryCommandTree
|_Parameters
|_Query : Collection{Record['C1'=Edm.Int32, 'ProductID'=Edm.Int32, 'ProductName'=Edm.String,
'CategoryName'=Edm.String, 'ShipCountry'=Edm.String, 'ProductID1'=Edm.Int32]}
  |_Project
    |_Input : 'Join4'
    | |_InnerJoin
    |   |_Left : 'Join1'
    |   | |_LeftOuterJoin
    |   |   |_Left : 'Extent1'
    |   |   | |_Scan : dbo.Products
    |   |   |_Right : 'Extent2'
    |   |   | |_Scan : dbo.Categories
    |   |   |_JoinCondition
    |   |     |_
    |   |       |_Var(Extent1).CategoryID
    |   |       |_=
    |   |       |_Var(Extent2).CategoryID
    |   |_Right : 'Join3'
    |   | |_LeftOuterJoin
    |   |   |_Left : 'Extent3'
    |   |   | |_Scan : dbo.OrderDetails
    |   |   |_Right : 'Join2'
    |   |   | |_LeftOuterJoin
    |   |   |   |_Left : 'Extent4'
    |   |   |   | |_Scan : dbo.Orders
    |   |   |   |_Right : 'Extent5'
    |   |   |   | |_Scan : dbo.InternationalOrders
    |   |   |   |_JoinCondition
    |   |   |     |_
    |   |   |       |_Var(Extent4).OrderID
    |   |   |       |_=
    |   |   |       |_Var(Extent5).OrderID
    |   |   |_JoinCondition
    |   |     |_
    |   |       |_Var(Extent3).OrderID
    |   |       |_=
    |   |       |_Var(Join2).Extent4.OrderID
    |   |_JoinCondition
    |     |_
    |       |_Var(Join1).Extent1.ProductID
    |       |_=
    |       |_Var(Join3).Extent3.ProductID
    |_Projection
      |_NewInstance : Record['C1'=Edm.Int32, 'ProductID'=Edm.Int32, 'ProductName'=Edm.String,
'CategoryName'=Edm.String, 'ShipCountry'=Edm.String, 'ProductID1'=Edm.Int32]
        |_Column : 'C1'
        | |_1
        |_Column : 'ProductID'
        | |_Var(Join4).Join1.Extent1.ProductID
        |_Column : 'ProductName'
        | |_Var(Join4).Join1.Extent1.ProductName
        |_Column : 'CategoryName'
        | |_Var(Join4).Join1.Extent2.CategoryName
        |_Column : 'ShipCountry'
        | |_Var(Join4).Join3.Join2.Extent4.ShipCountry
        |_Column : 'ProductID1'
          |_Var(Join4).Join3.Extent3.ProductID
```

En este tema se describe cómo traducir este árbol de comandos de salida en las siguientes instrucciones SQL.

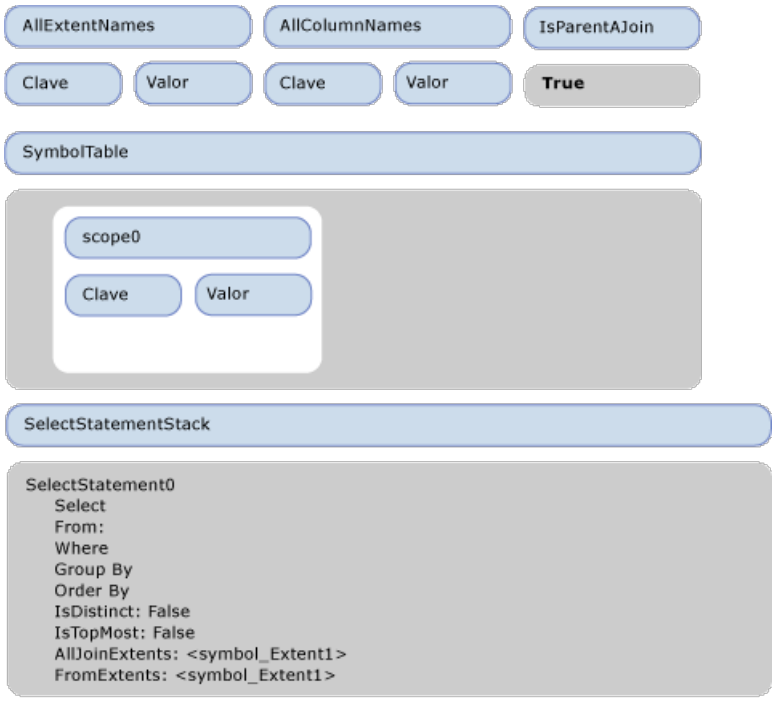
```
SELECT
1 AS [C1],
[Extent1].[ProductID] AS [ProductID],
[Extent1].[ProductName] AS [ProductName],
[Extent2].[CategoryName] AS [CategoryName],
[Join3].[ShipCountry] AS [ShipCountry],
[Join3].[ProductID] AS [ProductID1]
FROM   [dbo].[Products] AS [Extent1]
LEFT OUTER JOIN [dbo].[Categories] AS [Extent2] ON [Extent1].[CategoryID] = [Extent2].[CategoryID]
INNER JOIN
(SELECT [Extent3].[OrderID] AS [OrderID1], [Extent3].[ProductID] AS [ProductID], [Extent3].[UnitPrice] AS
[UnitPrice], [Extent3].[Quantity] AS [Quantity], [Extent3].[Discount] AS [Discount], [Join2].[OrderID2],
[Join2].[CustomerID], [Join2].[EmployeeID], [Join2].[OrderDate], [Join2].[RequiredDate], [Join2].
[ShippedDate], [Join2].[Freight], [Join2].[ShipName], [Join2].[ShipAddress], [Join2].[ShipCity], [Join2].
[ShipRegion], [Join2].[ShipPostalCode], [Join2].[ShipCountry], [Join2].[OrderID3], [Join2].
[CustomsDescription], [Join2].[ExciseTax]
FROM  [dbo].[OrderDetails] AS [Extent3]
LEFT OUTER JOIN
      (SELECT [Extent4].[OrderID] AS [OrderID2], [Extent4].[CustomerID] AS [CustomerID], [Extent4].
[EmployeeID] AS [EmployeeID], [Extent4].[OrderDate] AS [OrderDate], [Extent4].[RequiredDate] AS
[RequiredDate], [Extent4].[ShippedDate] AS [ShippedDate], [Extent4].[Freight] AS [Freight], [Extent4].
[ShipName] AS [ShipName], [Extent4].[ShipAddress] AS [ShipAddress], [Extent4].[ShipCity] AS [ShipCity],
[Extent4].[ShipRegion] AS [ShipRegion], [Extent4].[ShipPostalCode] AS [ShipPostalCode], [Extent4].
[ShipCountry] AS [ShipCountry], [Extent5].[OrderID] AS [OrderID3], [Extent5].[CustomsDescription] AS
[CustomsDescription], [Extent5].[ExciseTax] AS [ExciseTax]
FROM  [dbo].[Orders] AS [Extent4]
LEFT OUTER JOIN [dbo].[InternationalOrders] AS [Extent5] ON [Extent4].[OrderID] = [Extent5].[OrderID]
      ) AS [Join2] ON [Extent3].[OrderID] = [Join2].[OrderID2]
   ) AS [Join3] ON [Extent1].[ProductID] = [Join3].[ProductID]
```

Primera fase de la generación de SQL: visitar el árbol de expresiones

La siguiente figura muestra el estado vacío inicial del visitante. A lo largo de este tema, solo se muestran las propiedades pertinentes a la explicación del tutorial.



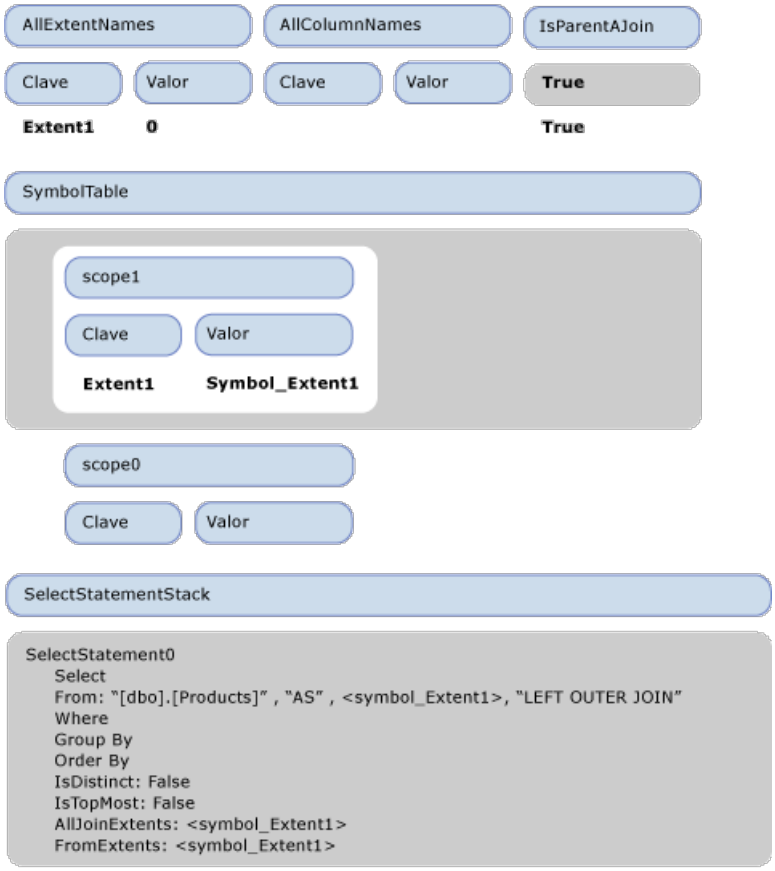
Cuando se visita el nodo Project, se llama a VisitInputExpression en su entrada (Join4), que activa la visita de Join4 por el método VisitJoinExpression. Dado que esta es la combinación de nivel superior, IsParentAJoin devuelve el valor false y se crea una nueva instrucción SqlSelectStatement (SelectStatement0) que se inserta en la pila de instrucciones SELECT. Asimismo, se introduce un nuevo ámbito (scope0) en la tabla de símbolos. Antes de visitar la primera entrada (izquierda) de la combinación, se inserta "true" en la pila de IsParentAJoin. Justo antes de visitar Join1, que es la entrada izquierda de Join4, el estado del visitante es el que se muestra en la figura siguiente.



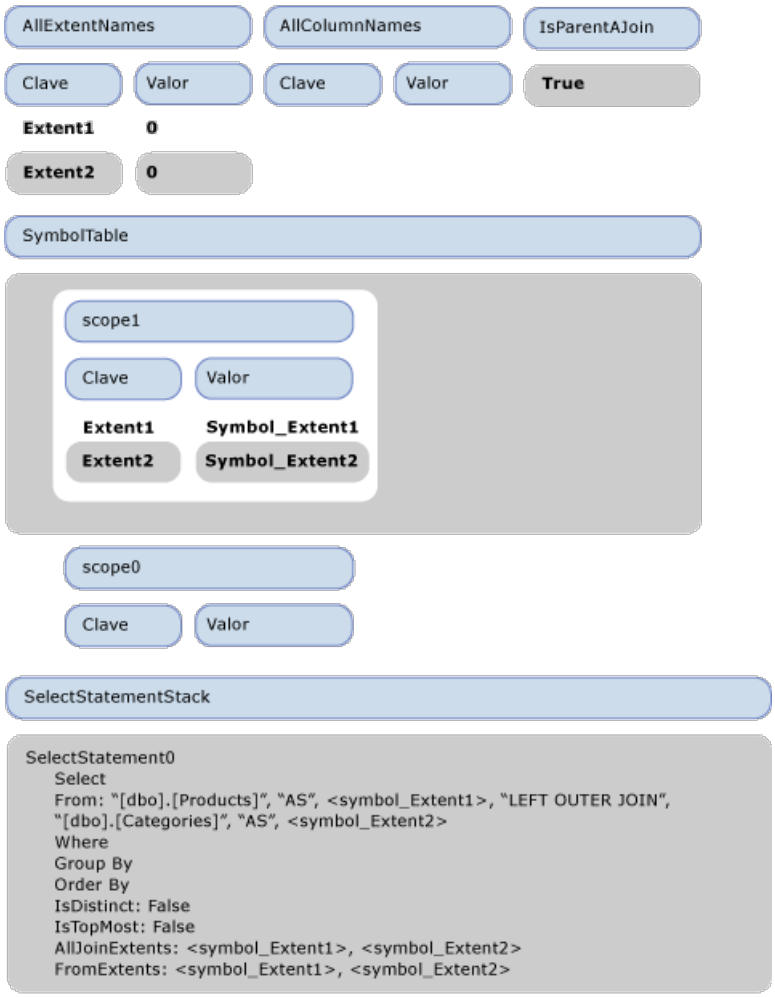
Cuando el método de visita de combinación se invoca en Join4, IsParentAJoin es "true", por lo que reutiliza la instrucción de selección SelectStatement0 actual. Se introduce un nuevo ámbito (scope1). Antes de visitar su elemento secundario izquierdo, Extent1, se inserta otro valor "true" en la pila de IsParentAJoin.

Cuando se visita Extent1, dado que IsParentAJoin devuelve "true", devuelve un objeto SqlBuilder que contiene "[dbo].[Products]". El control vuelve al método que visita Join4. Se extrae una entrada de IsParentAJoin y se llama a ProcessJoinInputResult, que anexa el resultado de la visita a Extent1 a la cláusula From de SelectStatement0. Se crea un nuevo símbolo FROM, symbol_Extent1, para el nombre de enlace de entrada "Extent1" y se agrega a FromExtents de SelectStatement0. Asimismo, se anexan "As" y symbol_Extent1 a la cláusula FROM. Se agrega una nueva entrada a AllExtentNames para "Extent1" con el valor de 0. Se agrega una nueva entrada al ámbito actual en la tabla de símbolos para asociar "Extent1" a su símbolo symbol_Extent1. Symbol_Extent1 también se agrega a AllJoinExtents de SqlSelectStatement.

Antes de que se visite la entrada derecha de Join1, se agrega "LEFT OUTER JOIN" a la cláusula From de SelectStatement0. Dado que la entrada derecha es una expresión Scan, se inserta de nuevo "true" en la pila de IsParentAJoin. El estado antes de visitar la entrada derecha es el que se muestra en la figura siguiente.



La entrada derecha se procesa de la misma manera que la entrada izquierda. El estado después de visitar la entrada derecha es el que se muestra en la figura siguiente.



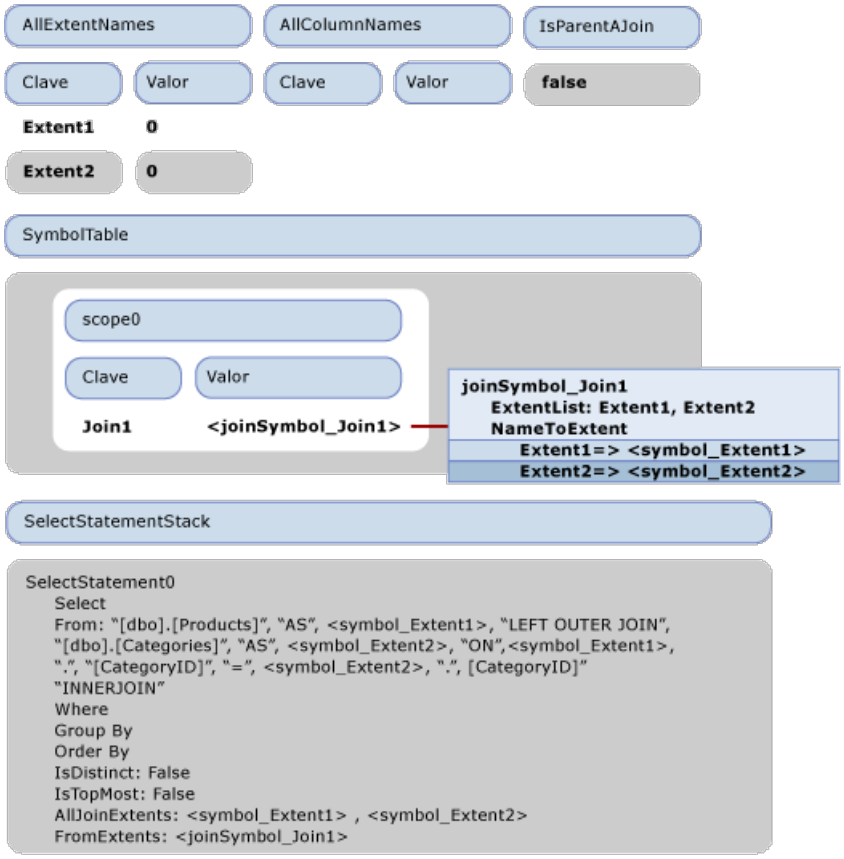
A continuación, se inserta "false" en la pila de IsParentAJoin y se procesa la condición de combinación

Var(Extent1).CategoryID == Var(Extent2).CategoryID. Var (Extent1) se resuelve en <symbol_Extent1 > después de una búsqueda en la tabla de símbolos. Dado que la instancia se resuelve en un símbolo simple, como resultado del procesamiento de var (Extent1). CategoryID, un SqlBuilder con <symbol1 > ". CategoryID ". De igual forma se procesa el otro lado de la comparación. El resultado de la visita de la condición de combinación se anexa a la cláusula FROM de SelectStatement1 y el valor "false" se extrae de la pila de IsParentAJoin.

Con esto, Join1 se ha procesado completamente y se ha extraído un ámbito de la tabla de símbolos.

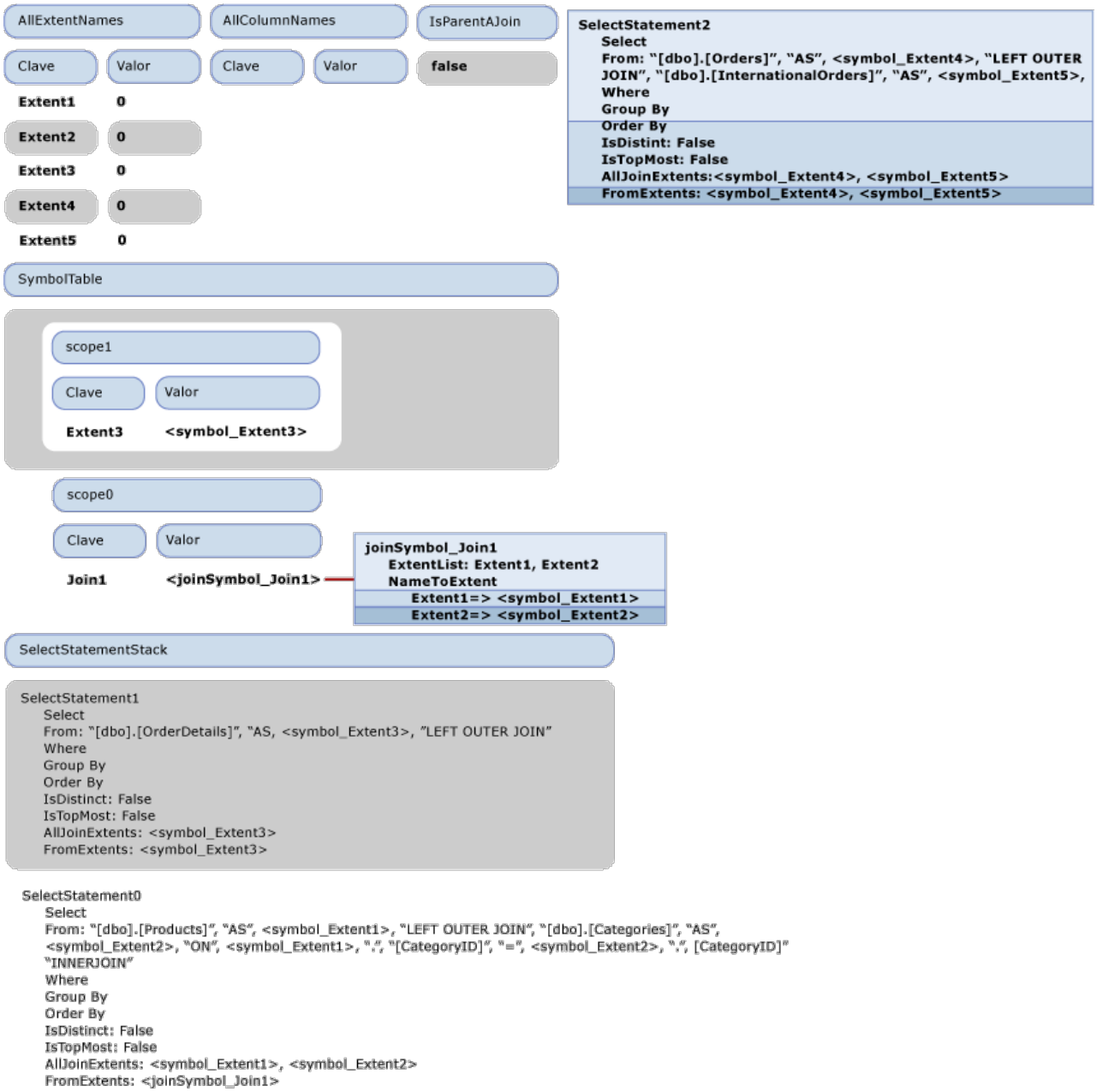
El control vuelve a procesar Join4, el elemento primario de Join1. Dado que el elemento secundario reutilizó la instrucción SELECT, las extensiones Join1 se reemplazan por un único símbolo de combinación <joinSymbol_Join1 >. También se agrega una nueva entrada a la tabla de símbolos para asociar Join1 con <joinSymbol_Join1 >.

El nodo siguiente que se va a procesar es Join3, el segundo elemento secundario de Join4. Como es un elemento secundario derecho, se inserta "false" en la pila de IsParentAJoin. En la figura siguiente se muestra el estado del visitante en este punto.



Para Join3, IsParentAJoin devuelve "false" y necesita iniciar una nueva instrucción SqlSelectStatement (SelectStatement1) e insertarla en la pila. El procesamiento continúa de la misma forma que en las combinaciones anteriores, se inserta un nuevo ámbito en la pila y se procesan los elementos secundarios. El elemento secundario izquierdo es una extensión (Extent3) y el elemento secundario derecho es una combinación (Join2) que también necesita iniciar una nueva instrucción SqlSelectStatement: SelectStatement2. Los elementos secundarios de Join2 también son extensiones y se agregan en SelectStatement2.

En la siguiente figura se muestra el estado del visitante justo después de visitar Join2, pero antes de que se realice su procesamiento posterior (ProcessJoinInputResult):



En la figura anterior, SelectStatement2 se muestra como flotante porque se extrajo de la pila, pero todavía no se ha procesado en el elemento primario. Necesita agregarse a la parte FROM del elemento primario, pero no es una instrucción SQL completa sin una cláusula SELECT. Así que, en este punto, las columnas predeterminadas (todas las columnas generadas por sus entradas) se agregan a la lista de selección mediante el método AddDefaultColumns. AddDefaultColumns recorre en iteración los símbolos de FromExtents y para cada símbolo agrega todas las columnas introducidas en el ámbito. Para un símbolo simple, examina el tipo de símbolo para recuperar todas sus propiedades que se van a agregar. También rellena el diccionario de AllColumnNames con los nombres de columna. La instrucción SelectStatement2 completa se anexa a la cláusula FROM de SelectStatement1.

A continuación, se crea un nuevo símbolo de combinación para representar Join2, se marca como combinación anidada y se agrega a AllJoinExtents de SelectStatement1 y a la tabla de símbolos. Ahora es necesario procesar la condición de combinación de Join3, Var(Extent3).OrderID = Var(Join2).Extent4.OrderID. El procesamiento del lado izquierdo es similar a la condición de combinación de Join1. Sin embargo, el procesamiento del lado derecho "Var(Join2).Extent4.OrderID" es diferente porque es necesario reducir la combinación.

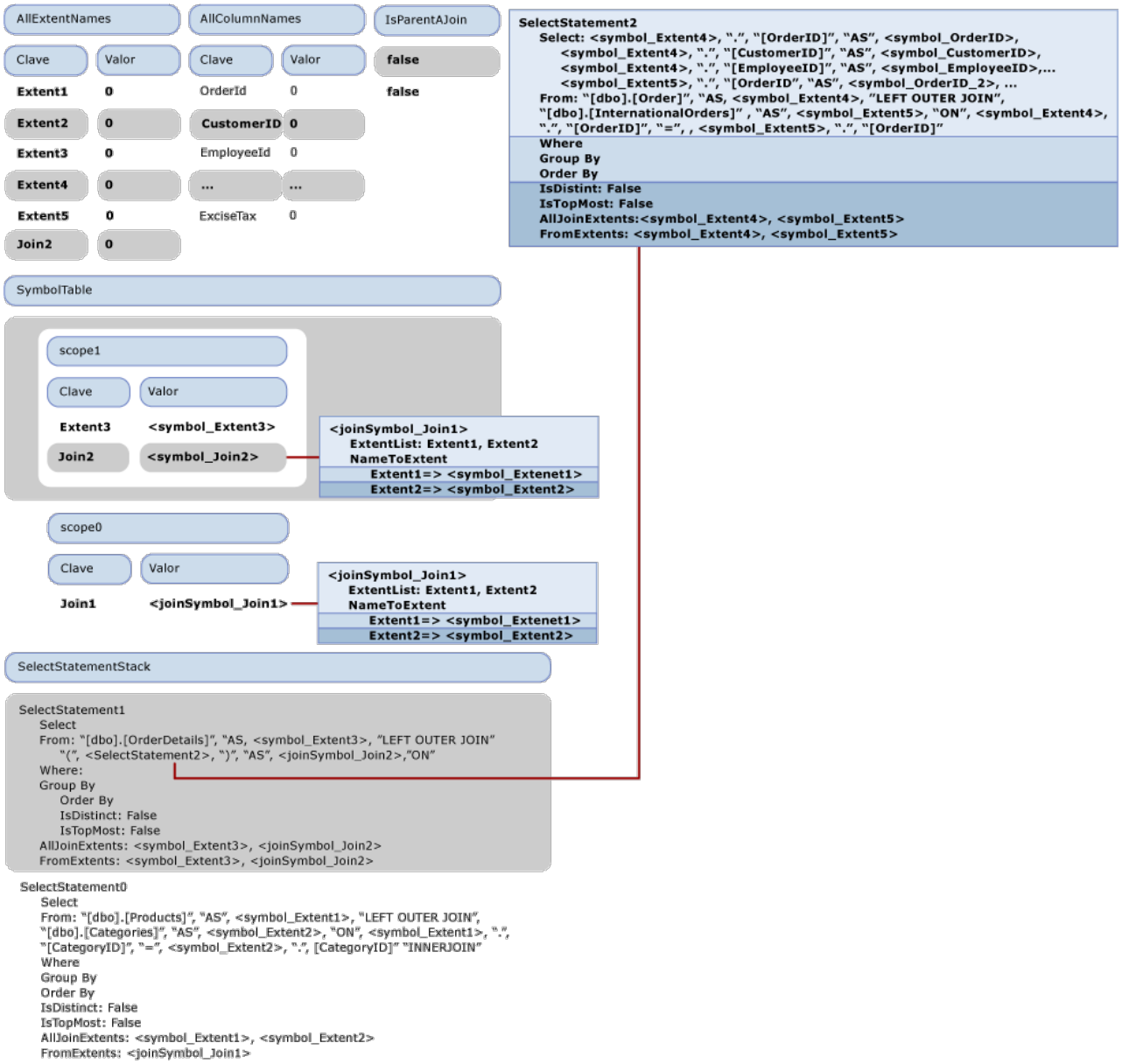
La figura siguiente muestra el estado del visitante justo antes de procesar la expresión DbPropertyExpression "Var(Join2).Extent4.OrderID".

Observe cómo se visita "Var(Join2).Extent4.OrderID". En primer lugar, se visita la propiedad de instancia "Var(Join2).Extent4", que es otra expresión DbPropertyExpression y visita primero su instancia "Var(Join2)". En el

ámbito de la parte superior de la tabla de símbolos, "Join2" se resuelve como <joinSymbol_join2 >. En el método de visita en el que DbPropertyExpression procesa "Var(Join2).Extent4", observe que se devuelve un símbolo de combinación al visitar la instancia y que es preciso quitar información de estructura jerárquica.

Dado que se trata de una combinación anidada, se busca la propiedad "Extent4" en el Diccionario de NameToExtent del símbolo de combinación, se resuelve en <symbol_Extent4 > y se devuelve un nuevo SymbolPair (<joinSymbol_join2 >, <symbol_Extent4 >). Dado que se devuelve un par de símbolos desde el procesamiento de la instancia de "var (Join2)". Extent4. OrderID ", la propiedad" OrderID "se resuelve desde el valor columnpart de ese par de símbolos (<symbol_Extent4 >), que tiene una lista de las columnas de la extensión que representa. Por lo tanto, "var (Join2). Extent4. OrderID "se resuelve en {<joinSymbol_Join2 >,". "<symbol_OrderID >}.

De igual forma se procesa la condición de combinación de Join4. El control vuelve al método VisitInputExpression que procesó el proyecto de nivel superior. Si se examina FromExtents de la instrucción SelectStatement0 devuelta, la entrada se identifica como una combinación, se quitan las extensiones originales y se reemplazan con una nueva extensión que únicamente incluye el símbolo de combinación. La tabla de símbolos también se actualiza y, a continuación, se procesa la parte de la proyección del proyecto. La resolución de las propiedades y la reducción de la extensión de la combinación se realizan como se ha descrito anteriormente.



Por último, se genera la siguiente instrucción SqlSelectStatement:

```
SELECT:
  "1", " AS ", "[C1]",
  <symbol_Extent1>, ".", "[ProductID]", " AS ", "[ProductID]",
  <symbol_Extent1>, ".", "[ProductName]", " AS ", "[ProductName]",
  <symbol_Extent2>, ".", "[CategoryName]", " AS ", "[CategoryName]",
  <joinSymbol_Join3>, ".", <symbol_ShipCountry>, " AS ", "[ShipCountry]",
  <joinSymbol_Join3>, ".", <symbol_ProductID>, " AS ", "[ProductID1]"
FROM: "[dbo].[Products]", " AS ", <symbol_Extent1>,
       "LEFT OUTER JOIN ""[dbo].[Categories]", " AS ", <symbol_Extent2>, " ON ", <symbol_Extent1>, ".", "
[CategoryID]", " = ", <symbol_Extent2>, ".", "[CategoryID]",
       "INNER JOIN ",
       " (", SELECT:
          <symbol_Extent3>, ".", "[OrderID]", " AS ", <symbol_OrderID>, ",
             <symbol_Extent3>, ".", "[ProductID]", " AS ", <symbol_ProductID>, ...,
        <joinSymbol_Join2>, ".", <symbol_OrderID_2>, ", ",
          <joinSymbol_Join2>, ".", <symbol_CustomerID>, ....,
       <joinSymbol_Join2>, ".", <symbol_OrderID_3>,
<joinSymbol_Join2>, ".", <symbol_CustomsDescription>,
<joinSymbol_Join2>, ".", <symbol_ExciseTax>
FROM: "[dbo].[OrderDetails]", " AS ", <symbol_Extent3>,
"LEFT OUTER JOIN ",
" (", SELECT:
<symbol_Extent4>, ".", "[OrderID]", " AS ", <symbol_OrderID_2>,
<symbol_Extent4>, ".", "[CustomerID]", " AS ", <symbol_CustomerID>, ...
<symbol_Extent5>, ".", "[OrderID]", " AS ", <symbol_OrderID_3>,
<symbol_Extent5>, ".", "[CustomsDescription]", " AS ", <symbol_CustomsDescription>,
<symbol_Extent5>, ".", "[ExciseTax]", " AS ", <symbol_ExciseTax>
FROM: "[dbo].[Orders]", " AS ", <symbol_Extent4>,
"LEFT OUTER JOIN ", , "[dbo].[InternationalOrders]", " AS ", <symbol_Extent5>,
" ON ", <symbol_Extent4>, ".", "[OrderID]", " = ", , <symbol_Extent5>, ".", "[OrderID]"
" )", " AS ", <joinSymbol_Join2>, " ON ", , , <symbol_Extent3>, ".", "[OrderID]", " = ", , <joinSymbol_Join2>,
".", <symbol_OrderID_2>
" )", " AS ", <joinSymbol_Join3>, " ON ", , , <symbol_Extent1>, ".", "[ProductID]", " = ", ,
<joinSymbol_Join3>, ".", <symbol_ProductID>
```

Segunda fase de la generación de SQL: Generar el comando String

La segunda fase genera los nombres reales de los símbolos; únicamente nos centraremos en los símbolos que representan las columnas denominadas "OrderID", ya que en este caso es necesario resolver un conflicto. Estas columnas se resaltan en la instrucción SqlSelectStatement. Tenga en cuenta que los sufijos utilizados en la figura solo pretenden destacar que se trata de instancias diferentes y no representan ningún nuevo nombre, ya que en esta fase todavía no se han asignado sus nombres finales (posiblemente diferentes de los nombres originales).

El primer símbolo encontrado al que se debe cambiar el nombre es <symbol_OrderID >. Su nuevo nombre se asigna como "OrderID1", 1 se marca como el último sufijo utilizado para "OrderID" y el símbolo se marca como símbolo que no necesita cambio de nombre. A continuación, se encuentra el primer uso de <symbol_OrderID_2 >. Se cambia el nombre para utilizar el siguiente sufijo disponible ("OrderID2") y de nuevo se marca como símbolo que no necesita cambio de nombre, para que la próxima vez que se utilice no se cambie el nombre. Esto se hace solo <> symbol_OrderID_3.

Al final de la segunda fase, se genera la instrucción SQL final.

Vea también

- Generación de SQL en el proveedor de ejemplo

Generar SQL de modificación

29/10/2019 • 16 minutes to read • Edit Online

En esta sección se describe cómo desarrollar un módulo de generación de SQL de modificación para el proveedor (de bases de datos conformes a SQL:1999). Este módulo es responsable de la conversión de un árbol de comandos de modificación en las instrucciones INSERT, UPDATE o DELETE de SQL adecuadas.

Para obtener información sobre la generación de SQL para las instrucciones SELECT, vea generación de SQL.

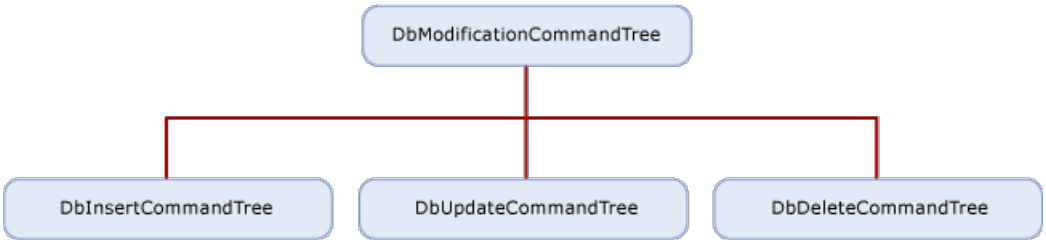
Información general sobre los árboles de comandos de modificación

El módulo de generación de SQL de modificación genera instrucciones SQL de modificación específicas de la base de datos en función de un DbModificationCommandTree de entrada determinado.

DbModificationCommandTree es una representación del modelo de objetos de una operación DML de modificación (una operación de inserción, actualización o eliminación), que hereda de DbCommandTree. Hay tres implementaciones de DbModificationCommandTree:

- DbInsertCommandTree
- DbUpdateCommandTree
- DbDeleteCommandTree

DbModificationCommandTree y sus implementaciones generadas por el Entity Framework siempre representan una operación de una sola fila. En esta sección se describen estos tipos con sus restricciones en .NET Framework versión 3.5.



DbModificationCommandTree tiene una propiedad de destino que representa el conjunto de destinos para la operación de modificación. La propiedad Expression del destino, que define el conjunto de entrada, siempre es DbScanExpression. Un DbScanExpression puede representar una tabla o una vista, o un conjunto de datos definido con una consulta si la propiedad de metadatos "definiendo consulta" de su destino no es NULL.

Una expresión DbScanExpression que representa una consulta solo puede alcanzar un proveedor como destino de la modificación si el conjunto se definió mediante una consulta de definición del modelo pero no se proporcionó ninguna función para la operación de modificación correspondiente. Es posible que los proveedores no puedan admitir este tipo de escenario (por ejemplo, SqlClient no puede).

DbInsertCommandTree representa una operación de inserción de una sola fila expresada como un árbol de comandos.

```
public sealed class DbInsertCommandTree : DbModificationCommandTree {
   public IList<DbModificationClause> SetClauses { get }
   public DbExpression Returning { get }
}
```

DbUpdateCommandTree representa una operación de actualización de una sola fila expresada como un árbol de comandos.

DbDeleteCommandTree representa una operación de eliminación de una sola fila expresada como un árbol de comandos.

Restricciones en las propiedades del árbol de comandos de modificación

La información y restricciones siguientes se aplican a las propiedades del árbol de comandos de modificación.

Returning en DbInsertCommandTree y DbUpdateCommandTree

Cuando no es NULL, Returning indica que el comando devuelve un lector. De lo contrario, el comando debe devolver un valor escalar que indique el número de filas afectadas (insertadas o actualizadas).

El valor de Returning especifica una proyección de resultados que se van a devolver en función de la fila insertada o actualizada. Solo puede ser de tipo DbNewInstanceExpression que representa una fila, y cada uno de sus argumentos es DbPropertyExpression en una expresión DbVariableReferenceExpression que representa una referencia al destino del árbol DbModificationCommandTree correspondiente. Las propiedades representadas por las expresiones DbPropertyExpressions utilizadas en la propiedad Returning siempre son valores calculados o generados por el almacén. En DbInsertCommandTree, Returning no es NULL cuando al menos una propiedad de la tabla en la que se inserta la fila se especifica como calculada o generada por el almacén (se marca como StoreGeneratedPattern.Identity o StoreGeneratedPattern.Computed en ssdl). En DbUpdateCommandTrees, Returning no es NULL cuando al menos una propiedad de la tabla en la que se actualiza la fila se especifica como calculada en el almacén (se marca como StoreGeneratedPattern.Computed en ssdl).

SetClauses en DbInsertCommandTree y DbUpdateCommandTree

SetClauses especifica la lista de cláusulas de inserción o actualización que definen la operación de inserción o actualización.

Los elementos de la lista se especifican como tipo DbModificationClause, que especifica una sola cláusula en una operación de modificación de inserción o actualización. DbSetClause hereda de DbModificationClause y especifica la cláusula en una operación de modificación que establece el valor de una propiedad. A partir de la versión 3,5 del .NET Framework, todos los elementos de SetClauses son del tipo SetClause.

Property especifica la propiedad que se debe actualizar. Siempre es DbPropertyExpression en DbVariableReferenceExpression, que representa una referencia al destino del árbol DbModificationCommandTree correspondiente.

Value especifica el nuevo valor con el que se actualiza la propiedad. Es de tipo DbConstantExpression o DbNullExpression.

Predicate en DbUpdateCommandTree y DbDeleteCommandTree

Predicate especifica el predicado que se usa para determinar qué miembros de la colección de destino se deben actualizar o eliminar. Es un árbol de expresión generado del siguiente subconjunto de DbExpressions:

- DbComparisonExpression of Kind es igual a, donde el elemento secundario Right es DbPropertyExpression como Restricted a continuación y el elemento secundario Left a DbConstantExpression.
- DbConstantExpression
- DbIsNullExpression a través de DbPropertyExpression como restringido a continuación
- DbPropertyExpression en DbVariableReferenceExpression, que representa una referencia al destino del árbol DbModificationCommandTree correspondiente.
- DbAndExpression
- DbNotExpression
- DbOrExpression

Generación de SQL de modificación en el proveedor de ejemplo

En el Entity Framework proveedor de ejemplo se muestran los componentes de los proveedores de datos de ADO.net que admiten el Entity Framework. Tiene como destino una base de datos de SQL Server 2005 y se implementa como un contenedor en el proveedor de datos ADO.NET 2.0 System.Data.SqlClient.

El módulo de generación de SQL de modificación del proveedor de ejemplo (situado en el archivo SQL Generation\DmlSqlGenerator.cs) toma un árbol DbModificationCommandTree como entrada y genera una única instrucción SQL de modificación posiblemente seguida por una instrucción SELECT para devolver un lector si se especifica en DbModificationCommandTree. Observe que la base de datos de SQL Server de destino afecta a la forma de los comandos generados.

clases del asistente: ExpressionTranslator

ExpressionTranslator actúa como un traductor ligero común para todas las propiedades del árbol de comandos de modificación de tipo DbExpression. Únicamente admite la traducción de los tipos de expresión a los que están restringidas las propiedades del árbol de comandos de modificación y se genera con las restricciones determinadas en mente.

La siguiente información describe la visita de tipos de expresión específicos (se omiten los nodos con traducciones triviales).

DbComparisonExpression

Cuando ExpressionTranslator se construye con preserveMemberValues = true y la constante de la derecha es DbConstantExpression (en lugar de DbNullExpression), asocia el operando izquierdo (DbPropertyExpressions) a DbConstantExpression. Se utiliza eso si es necesario generar una instrucción Select devuelta para identificar la fila afectada.

DbConstantExpression

Se crea un parámetro para cada constante visitada.

DbPropertyExpression

Dado que la instancia de DbPropertyExpression siempre representa la tabla de entrada, a menos que la generación haya creado un alias (lo que solo sucede en escenarios de actualización cuando se utiliza una variable de tabla), no es necesario especificar ningún alias para la entrada; la traducción tiene como valor predeterminado el nombre de propiedad.

Generar un comando SQL de inserción

Para una implementación DbInsertCommandTree determinada en el proveedor de ejemplo, el comando de inserción generado sigue una de las dos plantillas de inserción siguientes.

La primera plantilla incluye un comando para realizar la inserción dados los valores de la lista de SetClauses y una instrucción SELECT para devolver las propiedades especificadas en la propiedad Returning para la fila insertada si la propiedad Returning no es NULL. El elemento de predicado "@@ROWCOUNT > 0" es true si se insertó una fila. El elemento de predicado "keyMemberI | = keyValueI SCOPE_IDENTITY ()" toma la forma "keyMemberI = SCOPE_IDENTITY ()" solo si keyMemberI es una clave generada por el almacén, ya que SCOPE_IDENTITY () devuelve el último valor de identidad insertado en una identidad (columna generada por el almacén).

```
-- first insert Template
INSERT <target>   [ (setClauseProperty0, .. setClausePropertyN)]
VALUES (setClauseValue0, .. setClauseValueN) |  DEFAULT VALUES

[SELECT <returning>
 FROM <target>
 WHERE @@ROWCOUNT > 0 AND keyMember0 = keyValue0 AND .. keyMemberI =  keyValueI | scope_identity()  .. AND
keyMemberN = keyValueN]
```

La segunda plantilla es necesaria si la inserción especifica la inserción de una fila cuya clave principal es generada por el almacén pero no es de tipo entero y, por tanto, no se puede usar con scope_identity(). También se usa si existe una clave compuesta generada por el almacén.

```
-- second insert template
DECLARE @generated_keys TABLE [(keyMember0, … keyMemberN)

INSERT <target>   [ (setClauseProperty0, .. setClausePropertyN)]
 OUTPUT inserted.KeyMember0, …, inserted.KeyMemberN INTO @generated_keys
 VALUES (setClauseValue0, .. setClauseValueN) |  DEFAULT VALUES

[SELECT <returning_over_t>
 FROM @generated_keys  AS g
JOIN <target> AS t ON g.KeyMember0 = t.KeyMember0 AND … g.KeyMemberN = t.KeyMemberN
 WHERE @@ROWCOUNT > 0
```

A continuación se muestra un ejemplo que utiliza el modelo incluido con el proveedor de ejemplo. Genera un comando de inserción a partir de una implementación DbInsertCommandTree.

El siguiente código inserta una categoría:

```
using (NorthwindEntities northwindContext = new NorthwindEntities()) {
   Category c = new Category();
   c.CategoryName = "Test Category";
   c.Description = "A new category for testing";
   northwindContext.AddObject("Categories", c);
   northwindContext.SaveChanges();
}
```

Este código genera el siguiente árbol de comandos, que se pasa al proveedor:

```
DbInsertCommandTree
|_Parameters
|_Target : 'target'
| |_Scan : dbo.Categories
|_SetClauses
| |_DbSetClause
| | |_Property
| | | |_Var(target).CategoryName
| | |_Value
| |   |_'Test Category'
| |_DbSetClause
| | |_Property
| | | |_Var(target).Description
| | |_Value
| |   |_'A new category for testing'
| |_DbSetClause
|   |_Property
|   | |_Var(target).Picture
|   |_Value
|     |_null
|_Returning
  |_NewInstance : Record['CategoryID'=Edm.Int32]
    |_Column : 'CategoryID'
      |_Var(target).CategoryID
```

El comando de almacén que el proveedor de ejemplo genera es la siguiente instrucción SQL:

```
insert [dbo].[Categories]([CategoryName], [Description], [Picture])
values (@p0, @p1, null)
select [CategoryID]
from [dbo].[Categories]
where @@ROWCOUNT > 0 and [CategoryID] = scope_identity()
```

Generar un comando SQL de actualización

Para una implementación DbUpdateCommandTree determinada, el comando de actualización generado se basa en la siguiente plantilla:

```
-- UPDATE Template
UPDATE <target>
SET setClauseProperty0 = setClauseValue0, .. setClausePropertyN = setClauseValueN  | @i = 0
WHERE <predicate>

[SELECT <returning>
 FROM <target>
 WHERE @@ROWCOUNT > 0 AND keyMember0 = keyValue0 AND .. keyMemberI =  keyValueI | scope_identity()  .. AND
keyMemberN = keyValueN]
```

La cláusula SET tiene la cláusula SET falsa ("@i = 0") solo si no se especifica ninguna cláusula SET. Esto permite garantizar que las columnas calculadas en el almacén se vuelven a calcular.

Únicamente si la propiedad Returning no es NULL, se genera una instrucción SELECT para devolver las propiedades especificadas en la propiedad Returning.

En el siguiente ejemplo se utiliza el modelo que se incluye con el proveedor de ejemplo para generar un comando de actualización.

El siguiente código de usuario actualiza una categoría:

```
using (NorthwindEntities northwindContext = new NorthwindEntities()) {
   Category c = northwindContext.Categories.Where(i => i.CategoryName == "Test Category").First();
   c.CategoryName = "New test name";
   northwindContext.SaveChanges();
}
```

Este código de usuario genera el siguiente árbol de comandos, que se pasa al proveedor:

```
DbUpdateCommandTree
|_Parameters
|_Target : 'target'
| |_Scan : dbo.Categories
|_SetClauses
| |_DbSetClause
|   |_Property
|   | |_Var(target).CategoryName
|   |_Value
|     |_'New test name'
|_Predicate
| |_
|   |_Var(target).CategoryID
|   |_=
|   |_10
|_Returning
```

El proveedor de ejemplo genera el siguiente comando de almacén:

```
update [dbo].[Categories]
set [CategoryName] = @p0
where ([CategoryID] = @p1)
```

Generar un comando SQL de eliminación

Para una implementación DbDeleteCommandTree determinada, el comando DELETE generado se basa en la siguiente plantilla:

```
-- DELETE Template
DELETE <target>
WHERE <predicate>
```

En el siguiente ejemplo se utiliza el modelo que se incluye con el proveedor de ejemplo para generar un comando de eliminación.

El siguiente código de usuario elimina una categoría:

```
using (NorthwindEntities northwindContext = new NorthwindEntities()) {
   Category c = northwindContext.Categories.Where(i => i.CategoryName == "New test name").First();
   northwindContext.DeleteObject(c);
   northwindContext.SaveChanges();
}
```

Este código de usuario genera el siguiente árbol de comandos, que se pasa al proveedor.

```
DbDeleteCommandTree
|_Parameters
|_Target : 'target'
| |_Scan : dbo.Categories
|_Predicate
  |_
    |_Var(target).CategoryID
    |_=
    |_10
```

El proveedor de ejemplo genera el siguiente comando de almacén:

```
delete [dbo].[Categories]
where ([CategoryID] = @p0)
```

Vea también

- Escritura de un proveedor de datos de Entity Framework

Especificación del manifiesto del proveedor

21/03/2020 • 15 minutes to read • Edit Online

En esta sección se explica cómo puede un proveedor de almacén de datos admitir los tipos y funciones del almacén de datos.

Servicios de entidad funciona con independencia de un proveedor de almacén de datos concreto aunque permite a un proveedor de datos definir explícitamente la forma en que los modelos, las asignaciones y las consultas interactuarán con un almacén de datos subyacente. Sin una capa de abstracción, Servicios de entidad solo podría seleccionarse como destino en un determinado almacén de datos o proveedor de datos.

Los tipos que admite el proveedor están directa o indirectamente admitidos por la base de datos subyacente. Estos tipos no son necesariamente los tipos de almacén exactos, pero los tipos que usa el proveedor para admitir Entity Framework. Los tipos de proveedor/almacén se describen en los términos de Entity Data Model (EDM).

Los tipos de parámetro y de valores devueltos para las funciones admitidas por el almacén de datos se especifican en términos de EDM.

Requisitos

Entity Framework y el almacén de datos deben poder pasar datos de un lado a otro en tipos conocidos sin pérdida de datos ni truncamiento.

Las herramientas deben ser capaces de cargar el manifiesto del proveedor en tiempo de diseño sin tener que abrir una conexión al almacén de datos.

Entity Framework distingue mayúsculas de minúsculas, pero es posible que el almacén de datos subyacente no lo sea. Cuando los artefactos de EDM (identificadores y nombres de tipo, por ejemplo) se definen y se usan en el manifiesto, deben usar la sensibilidad a mayúsculas y minúsculas de Entity Framework. Si en el manifiesto del proveedor aparecen elementos de almacén de datos que pueden distinguir entre mayúsculas y minúsculas, esa grafía debe mantenerse en el manifiesto del proveedor.

Entity Framework requiere un manifiesto de proveedor para todos los proveedores de datos. Si intenta usar un proveedor que no tiene un manifiesto de proveedor con Entity Framework, obtendrá un error.

En la tabla siguiente se describen los tipos de excepciones que Entity Framework produciría cuando surjan excepciones a través de la interacción del proveedor:

PROBLEMA	EXCEPCIÓN
El proveedor no admite GetProviderManifest en DbProviderServices.	ProviderIncompatibleException
Falta el manifiesto del proveedor: el proveedor devuelve `null` cuando se intenta recuperar el manifiesto del proveedor.	ProviderIncompatibleException
Manifiesto del proveedor no válido: el proveedor devuelve XML no válido cuando se intenta recuperar el manifiesto del proveedor.	ProviderIncompatibleException

Escenarios

Un proveedor debe admitir los siguientes escenarios:

Escribir un proveedor con asignación de tipos simétrica

Puede escribir un proveedor para Entity Framework donde cada tipo de almacén se asigna a un único tipo de EDM, independientemente de la dirección de asignación. En el caso de un tipo de proveedor que tenga una asignación simple que se corresponda con un tipo de EDM, puede utilizar una solución simétrica, porque el sistema de tipos es simple o coincide con los tipos de EDM.

Puede utilizar la simplicidad de su dominio y generar un manifiesto del proveedor declarativo estático.

Puede escribir un archivo XML que tenga dos secciones:

- Una lista de tipos de proveedor expresada en términos de "equivalente de EDM" de un tipo de almacén o función. Los tipos de almacén tienen tipos de EDM equivalentes. Las funciones de almacén tienen funciones de EDM correspondientes. Por ejemplo, varchar es un tipo de SQL Server, pero el tipo de EDM correspondiente es string.
- Una lista de funciones admitida por el proveedor donde los tipos de parámetro y de valores devueltos se expresen en términos de EDM.

Escribir un proveedor con asignación de tipos asimétrica

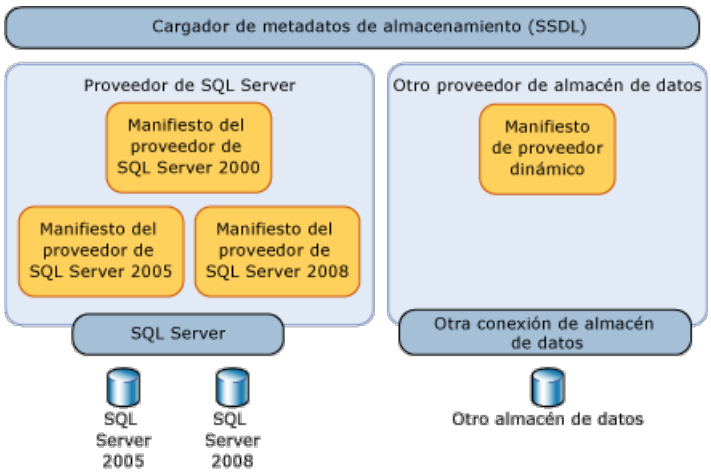
Al escribir un proveedor de almacén de datos para Entity Framework, la asignación de tipos de EDM a proveedor para algunos tipos puede ser diferente de la asignación de tipos de proveedor a EDM. Por ejemplo, la cadena PrimitiveTypeKind.String de EDM ilimitada se puede asignar a nvarchar(4000) en el proveedor, mientras que nvarchar(4000) se asigna a la cadena PrimitiveTypeKind.String(MaxLength=4000) de EDM.

Puede escribir un archivo XML que tenga dos secciones:

- Una lista de tipos de proveedor expresada en términos de EDM y defina la asignación para ambas direcciones: EDM a proveedor y proveedor a EDM.
- Una lista de funciones admitida por el proveedor donde los tipos de parámetro y de valores devueltos se expresen en términos de EDM.

Detectabilidad del manifiesto del proveedor

Varios tipos de componente utilizan indirectamente el manifiesto en Servicios de entidad (por ejemplo Tools o Query), pero los metadatos lo aprovechan de forma más directa con el uso del cargador de metadatos del almacén de datos.



Sin embargo, un proveedor determinado puede admitir almacenes diferentes o distintas versiones del mismo almacén. Por consiguiente, un proveedor debe notificar un manifiesto distinto para cada almacén de datos compatible.

Token del manifiesto del proveedor

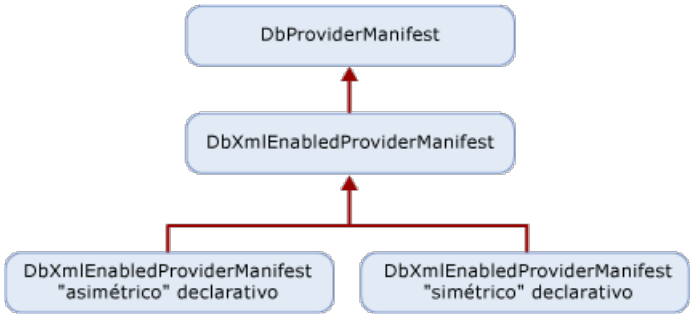
Cuando se abre una conexión al almacén de datos, el proveedor puede consultar información para devolver el manifiesto correcto. Esto puede no ser posible en escenarios sin conexión en los que no haya información de conexión disponible o cuando no se pueda conectar con el almacén. Identifique el manifiesto mediante el atributo `ProviderManifestToken` del elemento `Schema` en el archivo .ssdl. No hay ningún formato obligatorio para este atributo; el proveedor elige la información mínima necesaria para identificar un manifiesto sin abrir una conexión al almacén.

Por ejemplo:

```
<Schema Namespace="Northwind" Provider="System.Data.SqlClient" ProviderManifestToken="2005"
xmlns:edm="http://schemas.microsoft.com/ado/2006/04/edm/ssdl"
xmlns="http://schemas.microsoft.com/ado/2006/04/edm/ssdl">
```

Modelo de programación del manifiesto del proveedor

Los proveedores se derivan de DbXmlEnabledProviderManifest, lo que les permite especificar sus manifiestos mediante declaración. La siguiente ilustración muestra la jerarquía de clases de un proveedor:



API de detectabilidad

El cargador Metadatos de almacenamiento (StoreItemCollection) carga el manifiesto de proveedor, bien mediante una conexión al almacén de datos o utilizando un token de manifiesto del proveedor.

Utilizar una conexión al almacén de datos

Cuando la conexión del almacén de datos está disponible, llame DbProviderServices.GetProviderManifestToken para devolver el token que se pasa al GetProviderManifest método, que devuelve DbProviderManifest. Este método delega en la `GetDbProviderManifestToken` implementación del proveedor de .

```
public string GetProviderManifestToken(DbConnection connection);
public DbProviderManifest GetProviderManifest(string manifestToken);
```

Usar un token de manifiesto del proveedor

En el caso del escenario sin conexión, el token se toma de la representación SSDL. El SSDL le permite especificar un ProviderManifestToken (consulte elemento de esquema (SSDL) para obtener más información). Por ejemplo, si no se puede abrir una conexión, SSDL tiene un token de manifiesto del proveedor que especifica información sobre el manifiesto.

```
public DbProviderManifest GetProviderManifest(string manifestToken);
```

Esquema del manifiesto del proveedor

El esquema de información definido para cada proveedor contiene la información estática que van a utilizar los metadatos:

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema elementFormDefault="qualified"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://schemas.microsoft.com/ado/2006/04/edm/providermanifest"
   xmlns:pm="http://schemas.microsoft.com/ado/2006/04/edm/providermanifest">

  <xs:element name="ProviderManifest">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Types" type="pm:TTypes" minOccurs="1" maxOccurs="1" />
        <xs:element name="Functions" type="pm:TFunctions" minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
      <xs:attribute name="Namespace" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="TVersion">
    <xs:attribute name="Major" type="xs:int" use="required" />
    <xs:attribute name="Minor" type="xs:int" use="required" />
    <xs:attribute name="Build" type="xs:int" use="required" />
    <xs:attribute name="Revision" type="xs:int" use="required" />
  </xs:complexType>

  <xs:complexType name="TIntegerFacetDescription">
    <xs:attribute name="Minimum" type="xs:int" use="optional" />
    <xs:attribute name="Maximum" type="xs:int" use="optional" />
    <xs:attribute name="DefaultValue" type="xs:int" use="optional" />
    <xs:attribute name="Constant" type="xs:boolean" default="false" />
  </xs:complexType>

  <xs:complexType name="TBooleanFacetDescription">
    <xs:attribute name="DefaultValue" type="xs:boolean" use="optional" />
    <xs:attribute name="Constant" type="xs:boolean" default="true" />
  </xs:complexType>

  <xs:complexType name="TDateTimeFacetDescription">
    <xs:attribute name="Constant" type="xs:boolean" default="false" />
  </xs:complexType>

  <xs:complexType name="TFacetDescriptions">
    <xs:choice maxOccurs="unbounded">
      <xs:element name="Precision" minOccurs="0" maxOccurs="1" type="pm:TIntegerFacetDescription"/>
      <xs:element name="Scale" minOccurs="0" maxOccurs="1" type="pm:TIntegerFacetDescription"/>
      <xs:element name="MaxLength" minOccurs="0" maxOccurs="1" type="pm:TIntegerFacetDescription"/>
      <xs:element name="Unicode" minOccurs="0" maxOccurs="1" type="pm:TBooleanFacetDescription"/>
      <xs:element name="FixedLength" minOccurs="0" maxOccurs="1" type="pm:TBooleanFacetDescription"/>
    </xs:choice>
  </xs:complexType>

  <xs:complexType name="TType">
    <xs:sequence>
      <xs:element name="FacetDescriptions" type="pm:TFacetDescriptions" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="Name" type="xs:string" use="required"/>
    <xs:attribute name="PrimitiveTypeKind" type="pm:TPrimitiveTypeKind" use="required" />
  </xs:complexType>

  <xs:complexType name="TTypes">
    <xs:sequence>
      <xs:element name="Type" type="pm:TType" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <xs:attributeGroup name="TFacetAttribute">
    <xs:attribute name="Precision" type="xs:int" use="optional"/>
    <xs:attribute name="Scale" type="xs:int" use="optional"/>
    <xs:attribute name="MaxLength" type="xs:int" use="optional"/>
    <xs:attribute name="Unicode" type="xs:boolean" use="optional"/>
    <xs:attribute name="FixedLength" type="xs:boolean" use="optional"/>
  </xs:attributeGroup>
```

```
  <xs:complexType name="TFunctionParameter">
    <xs:attribute name="Name" type="xs:string" use="required" />
    <xs:attribute name="Type" type="xs:string" use="required" />
    <xs:attributeGroup ref="pm:TFacetAttribute" />
    <xs:attribute name="Mode" type="pm:TParameterDirection" use="required" />
  </xs:complexType>

  <xs:complexType name="TReturnType">
    <xs:attribute name="Type" type="xs:string" use="required" />
    <xs:attributeGroup ref="pm:TFacetAttribute" />
  </xs:complexType>

  <xs:complexType name="TFunction">
    <xs:choice minOccurs="0" maxOccurs ="unbounded">
      <xs:element name ="ReturnType" type="pm:TReturnType" minOccurs="0" maxOccurs="1" />
      <xs:element name="Parameter" type="pm:TFunctionParameter" minOccurs="0" maxOccurs="unbounded"/>
    </xs:choice>
    <xs:attribute name="Name" type="xs:string" use="required" />
    <xs:attribute name="Aggregate" type="xs:boolean" use="optional" />
    <xs:attribute name="BuiltIn" type="xs:boolean" use="optional" />
    <xs:attribute name="StoreFunctionName" type="xs:string" use="optional" />
    <xs:attribute name="NiladicFunction" type="xs:boolean" use="optional" />
    <xs:attribute name="ParameterTypeSemantics" type="pm:TParameterTypeSemantics" use="optional"
default="AllowImplicitConversion" />
  </xs:complexType>

  <xs:complexType name="TFunctions">
    <xs:sequence>
      <xs:element name="Function" type="pm:TFunction" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <xs:simpleType name="TPrimitiveTypeKind">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Binary"/>
      <xs:enumeration value="Boolean"/>
      <xs:enumeration value="Byte"/>
      <xs:enumeration value="Decimal"/>
      <xs:enumeration value="DateTime"/>
      <xs:enumeration value="Time"/>
      <xs:enumeration value="DateTimeOffset"/>
      <xs:enumeration value="Double"/>
      <xs:enumeration value="Guid"/>
      <xs:enumeration value="Single"/>
      <xs:enumeration value="SByte"/>
      <xs:enumeration value="Int16"/>
      <xs:enumeration value="Int32"/>
      <xs:enumeration value="Int64"/>
      <xs:enumeration value="String"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="TParameterDirection">
    <xs:restriction base="xs:string">
      <xs:enumeration value="In"/>
      <xs:enumeration value="Out"/>
      <xs:enumeration value="InOut"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="TParameterTypeSemantics">
    <xs:restriction base="xs:string">
      <xs:enumeration value="ExactMatchOnly" />
      <xs:enumeration value="AllowImplicitPromotion" />
      <xs:enumeration value="AllowImplicitConversion" />
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

Nodo Types

El nodo Types del manifiesto del proveedor contiene información sobre los tipos admitidos de forma nativa por el almacén de datos o a través del proveedor.

Nodo Type

Cada nodo Type define un tipo de proveedor en términos de EDM. El nodo Type describe el nombre del tipo de proveedor e información relacionada con el tipo de modelo al que está asignado, así como facetas que describen esa asignación de tipos.

Para expresar esta información de tipos en el manifiesto del proveedor, cada declaración de TypeInformation debe definir distintas descripciones de facetas para cada Type:

NOMBRE DEL ATRIBUTO	TIPO DE DATOS	OBLIGATORIO	VALOR PREDETERMINADO	DESCRIPCIÓN
Nombre	String	Sí	N/D	El nombre del tipo de datos específico del proveedor
PrimitiveTypeKind	PrimitiveTypeKind	Sí	N/D	Nombre de tipo EDM

Nodo Function

Cada nodo Function define una función única disponible a través del proveedor.

NOMBRE DEL ATRIBUTO	TIPO DE DATOS	OBLIGATORIO	VALOR PREDETERMINADO	DESCRIPCIÓN
Nombre	String	Sí	N/D	Identificador/nombre de la función
ReturnType	String	Sin	Void	El tipo de valor devuelto EDM de la función
Agregado	Boolean	Sin	False	True si la función es una función de agregado
BuiltIn	Boolean	Sin	True	True si la función está integrada en el almacén de datos
StoreFunctionName	String	Sin	<Name>	Nombre de la función en el almacén de datos. Permite un nivel de redirección de nombres de función.
NiladicFunction	Boolean	Sin	False	True si la función no requiere parámetros y se invoca sin ningún parámetro

NOMBRE DEL ATRIBUTO	TIPO DE DATOS	OBLIGATORIO	VALOR PREDETERMINADO	DESCRIPCIÓN
ParameterType Semántica	ParameterSemantics	Sin	AllowImplicit Conversión	Opción de cómo la canalización de la consulta debe tratar la sustitución de tipos de parámetro: - ExactMatchOnly - AllowImplicitPromotion - AllowImplicitConversion

Nodo Parameters

Cada función tiene una colección de uno o más nodos Parameter.

NOMBRE DEL ATRIBUTO	TIPO DE DATOS	OBLIGATORIO	VALOR PREDETERMINADO	DESCRIPCIÓN
Nombre	String	Sí	N/D	Identificador/nombre del parámetro.
Tipo	String	Sí	N/D	El tipo EDM del parámetro.
Mode	Parámetro Dirección	Sí	N/D	Dirección del parámetro: - en - fuera - inout

Atributo Namespace

Cada proveedor de almacén de datos debe definir un espacio de nombres o un grupo de espacios de nombres para la información definida en el manifiesto. Este espacio de nombres se puede utilizar en consultas de Entity SQL para resolver nombres de funciones y tipos. Por ejemplo: SqlServer. Ese espacio de nombres debe ser distinto del espacio de nombres canónico, EDM, definido por Servicios de entidad para que las consultas de Entity SQL admitan las funciones estándar.

Consulte también

- Escribir un proveedor de datos de Entity Framework

Consideraciones de desarrollo e implementación

23/10/2019 • 2 minutes to read • Edit Online

Los temas de esta sección tratan aspectos que hay que considerar cuando se desarrolla o implementa una aplicación basada en ADO.NET Entity Framework.

En esta sección

Consideraciones de seguridad

Consideraciones sobre el rendimiento

Consideraciones de migración

Consideraciones de implementación

Vea también

- ADO.NET Entity Framework
- Información general sobre Entity Framework
- Introducción
- ADO.NET Entity Data Model herramientas

Consideraciones de seguridad (Entity Framework)

20/02/2020 • 24 minutes to read • Edit Online

En este tema se describen las consideraciones de seguridad específicas para el desarrollo, la implementación y la ejecución de aplicaciones Entity Framework. También debe seguir las recomendaciones para crear aplicaciones de .NET Framework seguras. Para más información, consulte Introducción a la seguridad.

Consideraciones generales de seguridad

Las siguientes consideraciones de seguridad se aplican a todas las aplicaciones que utilizan el Entity Framework.

Usar solo proveedores de orígenes de datos de confianza.

Para comunicarse con el origen de datos, un proveedor debe hacer lo siguiente:

- Recibir la cadena de conexión del Entity Framework.
- Traducir el árbol de comandos al lenguaje de consultas nativo del origen de datos.
- Ensamblar y devolver los conjuntos de resultados.

Durante la operación de inicio de sesión, la información que se basa en la contraseña del usuario se pasa al servidor a través de las bibliotecas de red del origen de datos subyacente. Un proveedor malintencionado puede robar las credenciales del usuario, generar consultas malintencionadas o alterar el conjunto de resultados.

Cifrar la conexión para proteger los datos confidenciales.

El Entity Framework no controla directamente el cifrado de datos. Si los usuarios tienen acceso a los datos a través de una red pública, la aplicación debería establecer una conexión cifrada al origen de datos para aumentar la seguridad. Para obtener más información, consulte la documentación relacionada con la seguridad correspondiente al origen de datos. Para obtener una SQL Server origen de datos, vea cifrar conexiones a SQL Server.

Proteger la cadena de conexión.

La protección del acceso al origen de datos es uno de los objetivos más importantes a la hora de proteger una aplicación. Una cadena de conexión presenta una vulnerabilidad potencial si no se protege o si se construye incorrectamente. Al almacenar la información de conexión en texto sin formato o conservarla en la memoria, se pone en riesgo todo el sistema. A continuación se enumeran métodos recomendados para proteger las cadenas de conexión:

- Utilice la autenticación de Windows con un origen de datos de SQL Server.

  Al utilizar la autenticación de Windows para conectarse a un origen de datos de SQL Server, la cadena de conexión no contiene información de contraseñas ni del inicio de sesión.

- Cifre las secciones del archivo de configuración mediante una configuración protegida.

  ASP.NET incluye una característica denominada configuración protegida, que permite cifrar la información confidencial en un archivo de configuración. Si bien se ha diseñado principalmente para ASP.NET, la configuración protegida también puede usarse para cifrar secciones de los archivos de configuración en aplicaciones Windows. Para obtener una descripción detallada de las nuevas capacidades de configuración protegida, vea cifrar la información de configuración mediante la configuración protegida.

- Almacene las cadenas de conexión en archivos de configuración protegidos.

  Nunca debería incrustar las cadenas de conexión en el código fuente. Las cadenas de conexión también se pueden almacenar en archivos de configuración, lo que elimina la necesidad de incrustarlas en el código de

la aplicación. De forma predeterminada, el Asistente para Entity Data Model almacena las cadenas de conexión en el archivo de configuración de la aplicación. Debe proteger este archivo para evitar el acceso no autorizado.

- Utilice generadores de cadenas de conexión al crear dinámicamente las conexiones.

  Si debe construir las cadenas de conexión en tiempo de ejecución, utilice la clase EntityConnectionStringBuilder. Esta clase de generador de cadenas ayuda a evitar los ataques de inyección en las cadenas de conexión validando y anulando la información de entrada no válida. Para obtener más información, vea Cómo: compilar una cadena de conexión de EntityConnection. Utilice también la clase de generador de cadenas adecuada para construir la cadena de conexión del origen de datos que forma parte de la cadena de conexión de Entity Framework. Para obtener información sobre los generadores de cadenas de conexión para los proveedores de ADO.NET, consulte compiladores de cadenas de conexión.

Para más información, consulte Proteger la información de conexión.

No exponga EntityConnection a usuarios que no sean de confianza.

Un objeto EntityConnection expone la cadena de conexión de la conexión subyacente. Un usuario con acceso a un objeto EntityConnection también puede cambiar el ConnectionState de la conexión subyacente. La clase EntityConnection no es segura para la ejecución de subprocesos.

No pase las conexiones fuera del contexto de seguridad.

Una vez establecida una conexión, no debe pasarla fuera del contexto de seguridad. Por ejemplo, un subproceso con permiso para abrir una conexión no debería almacenar la conexión en una ubicación global. Si la conexión está disponible en una ubicación global, otro subproceso malintencionado puede utilizar la conexión abierta sin que se le haya concedido ese permiso explícitamente.

Sea consciente de que la información de inicio de sesión y de contraseñas puede estar visible en un volcado de memoria.

Cuando la información de contraseñas y del inicio de sesión del origen de datos se proporciona en la cadena de conexión, se mantiene en la memoria hasta que la recolección de elementos no utilizados reclama los recursos. Esto impide determinar el momento en que una cadena de contraseña deja de estar en la memoria. Si una aplicación se bloquea, un archivo de volcado de memoria puede contener información de seguridad importante y el usuario que ejecuta la aplicación y cualquier usuario con acceso administrativo al equipo puede ver dicho archivo. Utilice la autenticación de Windows para las conexiones con Microsoft SQL Server.

Conceda a los usuarios únicamente los permisos necesarios en el origen de datos.

Los administradores de orígenes de datos solo deberían conceder a los usuarios los permisos que necesitan. Aunque Entity SQL no admite las instrucciones DML que modifican datos, como INSERT, UPDATE o DELETE, los usuarios todavía pueden tener acceso a la conexión al origen de datos. Un usuario malintencionado podría utilizar esta conexión para ejecutar instrucciones DML en el lenguaje nativo del origen de datos.

Ejecute las aplicaciones con los permisos mínimos.

Cuando permite que una aplicación administrada se ejecute con permisos de plena confianza, el .NET Framework no limita el acceso de la aplicación al equipo. De esta forma se puede permitir que una vulnerabilidad de seguridad en la aplicación ponga en peligro a todo el sistema. Para utilizar la seguridad de acceso del código y otros mecanismos de seguridad en el .NET Framework, debe ejecutar las aplicaciones mediante permisos de confianza parcial y con el conjunto mínimo de permisos que se necesitan para permitir que la aplicación funcione. Los permisos de acceso del código siguientes son los permisos mínimos que necesita la aplicación Entity Framework:

- FileIOPermission: Write para abrir los archivos de metadatos especificados o PathDiscovery para buscar los archivos de metadatos en un directorio.
- ReflectionPermission: RestrictedMemberAccess para admitir consultas de LINQ to Entities.
- DistributedTransactionPermission: Unrestricted para darse de alta en una System.TransactionsTransaction.
- SecurityPermission: SerializationFormatter para serializar las excepciones utilizando la interfaz ISerializable.

- Permiso para abrir una conexión de base de datos y ejecutar comandos en la base de datos, como SqlClientPermission para una base de datos de SQL Server.

Para obtener más información, consulta Code Access Security and ADO.NET.

No instale aplicaciones que no sean de confianza.

El Entity Framework no aplica ningún permiso de seguridad e invocará cualquier código de objeto de datos proporcionado por el usuario en proceso, independientemente de si es de confianza o no. Asegúrese de que la autenticación y la autorización del cliente se llevan a cabo en el almacén de datos y en la aplicación.

Restrinja el acceso a todos los archivos de configuración.

Un administrador debe restringir el acceso de escritura a todos los archivos que especifican la configuración de una aplicación, incluidos enterprisesec. config, Security. config, Machine. conf y el archivo de configuración de la *aplicación* <> . exe. config.

El nombre invariable del proveedor es modificable en el archivo app. config. La aplicación cliente debe asumir la responsabilidad de tener acceso al proveedor subyacente a través del modelo de generador de proveedores estándar mediante un nombre seguro.

Restrinja los permisos a los archivos de asignación y de modelo.

Un administrador debe restringir el acceso de escritura a los archivos de asignación y de modelo (.edmx, .csdl, .ssdl y .msl) únicamente a los usuarios que modifican el modelo o las asignaciones. El Entity Framework solo requiere acceso de lectura a estos archivos en tiempo de ejecución. Además, un administrador debe restringir el acceso a los archivos de código fuente de la capa de objetos y la vista previamente compilada generados por las herramientas de Entity Data Model.

Consideraciones de seguridad para las consultas

Se deben tener en cuenta las consideraciones de seguridad siguientes cuando se consulta un modelo conceptual. Estas consideraciones se aplican a las consultas de Entity SQL que usan EntityClient y en las consultas de objetos que usan LINQ, los métodos del generador de consultas o Entity SQL.

Impida los ataques de inyección de SQL.

A menudo, las aplicaciones obtienen entradas externas (de un usuario o de otro agente externo) y realizan acciones en función de dichas entradas. Cualquier entrada derivada directa o indirectamente del usuario o de un agente externo puede inyectar contenido que aproveche la sintaxis del lenguaje de destino para realizar acciones no autorizadas. Cuando el lenguaje de destino es un Lenguaje de consulta estructurado (SQL), como Transact-SQL, esta manipulación se conoce como ataque de inyección de SQL. Un usuario malintencionado puede inyectar comandos directamente en la consulta y colocar una tabla de la base de datos, provocar un ataque de denegación de servicio o cambiar de alguna otra forma la naturaleza de la operación que se está realizando.

- Ataques de inyección de Entity SQL:

  Los ataques de inyección de SQL se pueden realizar en Entity SQL proporcionando entradas malintencionadas a los valores que se utilizan en un predicado de consulta y en los nombres de los parámetros. Para evitar el riesgo de inyección de SQL, nunca debería combinar los datos proporcionados por el usuario con el texto de comandos de Entity SQL.

  Las consultas de Entity SQL aceptan parámetros siempre que se aceptan literales. Se deben usar consultas parametrizadas en lugar de insertar literales directamente en la consulta procedentes de un agente externo. También debe considerar el uso de métodos del generador de consultas para construir Entity SQL de forma segura.

- Ataques de inyección de LINQ to Entities:

  Aunque la composición de consultas es posible en LINQ to Entities, se realiza a través de la API del modelo de objetos. A diferencia de las consultas de Entity SQL, las consultas de LINQ to Entities no se componen

mediante la manipulación o la concatenación de cadenas y no son susceptibles a ataques de inyección de SQL tradicionales.

Evite los conjuntos de resultados muy grandes.

Un conjunto de resultados muy grande podría hacer que el sistema cliente se cerrara si el cliente realizara operaciones que consumieran recursos en proporción al tamaño del conjunto de resultados. Los conjuntos de resultados inesperadamente grandes se pueden producir en las condiciones siguientes:

- En consultas con una base de datos grande que no incluyen las condiciones de filtro adecuadas.
- En consultas que crean combinaciones cartesianas en el servidor.
- En consultas de Entity SQL.

Al aceptar datos proporcionados por el usuario, debe asegurarse de que no puedan provocar que los conjuntos de resultados se vuelvan mayores de lo que el sistema puede administrar. También puede utilizar el método Take en LINQ to Entities o el operador Limit en Entity SQL para limitar el tamaño del conjunto de resultados.

Evitar devolver resultados de IQueryable al exponer métodos a autores de llamadas que pueden no ser de confianza.

Evite devolver tipos IQueryable<T> desde métodos expuestos a autores de llamadas que pueden no ser de confianza por las siguientes razones:

- Un consumidor de una consulta que expone un tipo IQueryable<T> podría llamar a métodos sobre el resultado que exponen datos seguros o aumenta el tamaño del conjunto de resultados. Por ejemplo, considere la siguiente firma de método:

  ```
  public IQueryable<Customer> GetCustomer(int customerId)
  ```

  Un consumidor de esta consulta podría llamar a `.Include("Orders")` sobre el `IQueryable<Customer>` devuelto para recuperar datos que la consulta no pretendía exponer. Esto se puede evitar cambiando el tipo de valor devuelto del método a IEnumerable<T> y llamando a un método (como `.ToList()`) que materialice los resultados.

- Puesto que las consultas IQueryable<T> se ejecutan al iterar sobre los resultados, un consumidor de una consulta que expone un tipo IQueryable<T> podría interceptar las excepciones que se producen. Las excepciones podrían contener información no destinada para el consumidor.

Consideraciones de seguridad para entidades

Al generar y trabajar con tipos de entidad se aplican las consideraciones de seguridad siguientes.

No comparta un ObjectContext a través de dominios de aplicación.

Al compartir un ObjectContext con más de un dominio de aplicación, se puede exponer información en la cadena de conexión. En su lugar, debería transferir objetos serializados o gráficos de objetos al otro dominio de aplicación y, a continuación, asociar esos objetos a ObjectContext en ese dominio de aplicación. Para obtener más información, vea serializar objetos.

Evite las infracciones de seguridad de los tipos.

Si se infringe la seguridad de tipos, el Entity Framework no puede garantizar la integridad de los datos de los objetos. Se podrían producir infracciones de seguridad de tipos si permite que las aplicaciones que no son de confianza se ejecuten con seguridad de acceso del código de plena confianza.

Controle las excepciones.

Obtenga acceso a los métodos y propiedades de un ObjectContext dentro de un bloque try-catch. La detección de excepciones evita que las excepciones no controladas expongan las entradas del ObjectStateManager o la información del modelo (tal como nombres de las tablas) a los usuarios de la aplicación.

Consideraciones de seguridad para las aplicaciones ASP.NET

Debe tener en cuenta lo siguiente al trabajar con rutas de acceso en aplicaciones de ASP.NET.

Compruebe si el host realiza comprobaciones de la ruta de acceso.

Cuando se usa la cadena de sustitución `|DataDirectory|` (entre barras verticales), ADO.NET comprueba que se admite la ruta de acceso resuelta. Por ejemplo, ".." no se admite detrás de `DataDirectory`. La misma comprobación para resolver el operador raíz de la aplicación web (`~`) se realiza mediante el proceso que hospeda ASP.NET. IIS realiza esta comprobación; sin embargo, los hosts que no sean IIS pueden no comprobar que la ruta de acceso resuelta se admite. Debe conocer el comportamiento del host en el que implementa una aplicación Entity Framework.

No haga suposiciones sobre los nombres de ruta resueltos.

Aunque los valores a los que se resuelve el operador raíz (`~`) y la cadena de sustitución `DataDirectory` deben permanecer constantes durante el tiempo de ejecución de la aplicación, el Entity Framework no impide que el host modifique estos valores.

Compruebe la longitud de la ruta de acceso antes de la implementación.

Antes de implementar una aplicación Entity Framework, debe asegurarse de que los valores del operador raíz (~) y `DataDirectory` cadena de sustitución no superen los límites de la longitud de la ruta de acceso en el sistema operativo. Los proveedores de datos ADO.NET no garantizan que la longitud de la ruta de acceso esté dentro de los límites válidos.

Consideraciones de seguridad de los metadatos de ADO.NET

Al generar y trabajar con los archivos de asignación y de modelo, se aplican las consideraciones de seguridad siguientes.

No exponga información confidencial a través del registro.

Los componentes del servicio de metadatos de ADO.NET no registran información privada. Si hay resultados que no se pueden devolver debido a las restricciones de acceso, los sistemas de administración de bases de datos y los sistemas de archivos no deberían devolver ningún resultado en lugar de generar una excepción que podría contener información confidencial.

No acepte objetos MetadataWorkspace de orígenes que no sean de confianza.

Las aplicaciones no deberían aceptar instancias de la clase MetadataWorkspace de orígenes que no sean de confianza. En su lugar, debería construir y rellenar explícitamente un área de trabajo de este tipo de origen.

Consulte también

- Proteger aplicaciones de ADO.NET
- Consideraciones de implementación
- Consideraciones de migración

Consideraciones sobre el rendimiento (Entity Framework)

02/07/2020 • 28 minutes to read • Edit Online

En este tema se describen las características de rendimiento de ADO.NET Entity Framework y se facilitan algunas consideraciones para ayudar a mejorar el rendimiento de las aplicaciones de Entity Framework.

Fases de la ejecución de consultas

Para entender mejor el rendimiento de las consultas en Entity Framework, resulta útil comprender las operaciones que se realizan cuando se ejecuta una consulta en un modelo conceptual y se devuelven datos como objetos. En la tabla siguiente se describe esta serie de operaciones.

OPERACIÓN	COSTO RELATIVO	FRECUENCIA	COMENTARIOS
Cargar los metadatos	Moderado	Una vez en cada dominio de aplicación.	Los metadatos de la asignación y del modelo usados por Entity Framework se cargan en una instancia de MetadataWorkspace. Estos metadatos se almacenan globalmente en caché y están disponibles para otras instancias de ObjectContext en el mismo dominio de aplicación.
Abrir la conexión de la base de datos	Moderado[1]	Según sea necesario.	Dado que una conexión abierta a la base de datos consume un recurso valioso, el Entity Framework abre y cierra la conexión de base de datos solo según sea necesario. También puede abrir explícitamente la conexión. Para obtener más información, consulte Administración de conexiones y transacciones.

OPERACIÓN	COSTO RELATIVO	FRECUENCIA	COMENTARIOS
Generar vistas	Alto	Una vez en cada dominio de aplicación. (Se pueden generar previamente).	Antes de que Entity Framework pueda ejecutar una consulta en un modelo conceptual o guardar los cambios realizados en el origen de datos, debe generar un conjunto de vistas de consulta locales para obtener acceso a la base de datos. Debido al alto costo de generar estas vistas, es posible generarlas previamente y agregarlas al proyecto en tiempo de diseño. Para obtener más información, vea Cómo: generar previamente vistas para mejorar el rendimiento de las consultas.
Preparar la consulta	Moderada[2]	Una vez para cada consulta única.	Incluye los costos para crear el comando de consulta, generar un árbol de comandos basado en los metadatos de asignación y del modelo, y definir la forma de los datos devueltos. Dado que tanto los comandos de consulta de Entity SQL como las consultas LINQ se almacenan en memoria caché, las ejecuciones posteriores de la misma consulta tardarán menos tiempo. Todavía puede usar consultas LINQ compiladas para reducir este costo en ejecuciones posteriores y las consultas compiladas pueden ser más eficaces que las consultas LINQ que se almacenan en memoria caché automáticamente. Para obtener más información, vea consultas compiladas (LINQ to Entities). Para obtener información general sobre la ejecución de consultas LINQ, vea LINQ to Entities. **Nota:** LINQ to Entities consultas que aplican el `Enumerable.Contains` operador a colecciones en memoria no se almacenan en caché automáticamente. Tampoco se permite parametrizar colecciones en memoria en consultas LINQ compiladas.

OPERACIÓN	COSTO RELATIVO	FRECUENCIA	COMENTARIOS
Ejecutar la consulta	Bajo[2]	Una vez para cada consulta.	El costo de ejecutar el comando en el origen de datos usando el proveedor de datos de ADO.NET. Dado que la mayoría de los orígenes de datos almacenan en la caché los planes de consulta, las posteriores ejecuciones de la misma consulta pueden tardar menos tiempo.
Cargar y validar los tipos	Bajo[3]	Una vez para cada instancia de ObjectContext.	Los tipos se cargan y se validan con los tipos definidos por el modelo conceptual.
Seguimiento	Bajo[3]	Una vez para cada objeto devuelto por una consulta. [4]	Si una consulta usa la opción de fusión mediante combinación NoTracking, esta fase no afecta al rendimiento. Si la consulta usa la opción de fusión mediante combinación AppendOnly, PreserveChanges o OverwriteChanges, el seguimiento de los resultados de la consulta se realiza en la instancia de ObjectStateManager. Se genera una instancia de EntityKey para cada objeto devuelto por la consulta al que se ha hecho un seguimiento y se usa para crear un objeto ObjectStateEntry en la instancia de ObjectStateManager. Si se encuentra un objeto ObjectStateEntry existente para la instancia de EntityKey, se devuelve dicho objeto. Si se usa la opción PreserveChanges o OverwriteChanges, el objeto se actualiza antes de ser devuelto. Para obtener más información, vea resolución de identidades, administración de Estados y Change Tracking.

OPERACIÓN	COSTO RELATIVO	FRECUENCIA	COMENTARIOS
Materializar los objetos	Moderada[3]	Una vez para cada objeto devuelto por una consulta. [4]	El proceso de leer el objeto DbDataReader devuelto, crear los objetos y establecer valores de propiedad basados en los valores de cada instancia de la clase DbDataRecord. Si el objeto ya existe en ObjectContext y la consulta usa las opciones de fusión mediante combinación AppendOnly o PreserveChanges, esta fase no afecta al rendimiento. Para obtener más información, vea resolución de identidades, administración de Estados y Change Tracking.

[1] cuando un proveedor de origen de datos implementa la agrupación de conexiones, el costo de abrir una conexión se distribuye por el grupo. El proveedor de datos .NET para SQL Server admite la agrupación de conexiones.

[2] los costos aumentan con mayor complejidad de las consultas.

[3] el costo total aumenta proporcionalmente al número de objetos devueltos por la consulta.

[4] esta sobrecarga no es necesaria para las consultas de EntityClient porque las consultas de EntityClient devuelven un EntityDataReader objeto en lugar de objetos. Para obtener más información, consulte Proveedor de EntityClient para Entity Framework.

Consideraciones adicionales

A continuación se muestran otras consideraciones que pueden afectar al rendimiento de las aplicaciones de Entity Framework.

Ejecución de la consulta

Dado que las consultas usan muchos recursos, considere detenidamente en qué punto del código y en qué equipo se va ejecutar una consulta.

Ejecución diferida frente a ejecución inmediata

Cuando se crea una instancia de ObjectQuery<T> o una consulta LINQ, la consulta no se puede ejecutar inmediatamente. La ejecución de la consulta se aplaza hasta que se necesiten los resultados, como por ejemplo, durante una enumeración `foreach` (C#) o `For Each` (Visual Basic) o cuando se asigna para rellenar una colección List<T>. La ejecución de la consulta comienza inmediatamente cuando se llama al método Execute de una ObjectQuery<T> o cuando se llama a un método LINQ que devuelve una consulta singleton, como First o Any. Para obtener más información, vea consultas de objetos y ejecución de consultas (LINQ to Entities).

Ejecución de consultas LINQ en el cliente

Aunque la ejecución de una consulta LINQ se produce en el equipo que hospeda el origen de datos, algunas partes de dicha consulta se pueden evaluar en el equipo cliente. Para obtener más información, vea la sección ejecución de la tienda de ejecución de consultas (LINQ to Entities).

Complejidad de la asignación y de la consulta

La complejidad de las consultas individuales y de la asignación en el modelo de entidades tendrá un efecto significativo en el rendimiento de las consultas.

Complejidad de la asignación

Los modelos con una complejidad superior a la de una asignación unívoca simple entre entidades del modelo conceptual y tablas del modelo de almacenamiento generan comandos más complejos.

Complejidad de la consulta

Las consultas que requieren un gran número de combinaciones en los comandos que se ejecutan en el origen de datos o que devuelven una gran cantidad de datos pueden afectar al rendimiento de las maneras siguientes:

- Las consultas realizadas en un modelo conceptual y que parecen sencillas pueden ocasionar la ejecución de consultas más complejas en el origen de datos. Esto se puede producir porque Entity Framework traduce una consulta en un modelo conceptual en una consulta equivalente en el origen de datos. Cuando un único conjunto de entidades en el modelo conceptual se asigna a varias tablas en el origen de datos, o cuando una relación entre entidades se asigna a una tabla de combinación, el comando de consulta ejecutado en la consulta del origen de datos puede requerir una o más combinaciones.

  > **NOTE**
  > Use el método ToTraceString de las clases ObjectQuery<T> o EntityCommand para ver los comandos que se ejecutan en el origen de datos para una consulta determinada. Para obtener más información, consulte Cómo: ver los comandos de la tienda.

- Las consultas de Entity SQL anidadas pueden crear combinaciones en el servidor y devolver un gran número de filas.

  El siguiente es un ejemplo de una consulta anidada en una cláusula de proyección:

  ```
  SELECT c, (SELECT c, (SELECT c FROM AdventureWorksModel.Vendor AS c  ) As Inner2
      FROM AdventureWorksModel.JobCandidate AS c  ) As Inner1
      FROM AdventureWorksModel.EmployeeDepartmentHistory AS c
  ```

  Además, provocan que la canalización de la consulta genere una consulta única con duplicación de objetos en las consultas anidadas. Debido a esto, una sola columna se puede duplicar varias veces. En algunas bases de datos, incluyendo SQL Server, esto puede ocasionar que la tabla TempDB crezca demasiado, lo que puede disminuir el rendimiento del servidor. Conviene tener cuidado al ejecutar consultas anidadas.

- Las consultas que devuelven una gran cantidad de datos pueden disminuir el rendimiento si el cliente está realizando operaciones que tienen un consumo de recursos proporcional al tamaño del conjunto de resultados. En casos como este, considere la posibilidad de limitar la cantidad de datos devuelta por la consulta. Para obtener más información, vea Cómo: paginar a través de los resultados de la consulta.

Los comandos generados automáticamente por Entity Framework pueden ser más complejos que los comandos similares escritos explícitamente por un desarrollador de bases de datos. Si necesita un control explícito sobre los comandos ejecutados en el origen de datos, considere la posibilidad de definir una asignación a una función con valores de tabla o a un procedimiento almacenado.

Relaciones

Para obtener un rendimiento óptimo de las consultas, debe definir las relaciones entre entidades como asociaciones en el modelo de entidades y también como relaciones lógicas en el origen de datos.

Rutas de la consulta

De forma predeterminada, al ejecutar una consulta ObjectQuery<T>, no se devuelven los objetos relacionados (aunque los propios objetos que representan las relaciones lo son). Puede cargar objetos relacionados de una de las tres maneras siguientes:

1. Establezca la ruta de acceso de la consulta antes de que se ejecute ObjectQuery<T>.
2. Llame al método `Load` sobre la propiedad de navegación expuesta por el objeto.

3. Establezca la opción LazyLoadingEnabled del ObjectContext en `true`. Tenga en cuenta que esto se hace automáticamente cuando se genera código de nivel de objeto con el Diseñador de Entity Data Model. Para obtener más información, vea información general sobre el código generado.

Cuando considere qué opción se debe usar, sea consciente de que hay una correlación entre el número de solicitudes a la base de datos y la cantidad de datos devueltos en una sola consulta. Para obtener más información, vea cargar objetos relacionados.

Usar las rutas de la consulta

Las rutas de la consulta definen el gráfico de los objetos devueltos por una consulta. Al definir la ruta de una consulta, solo se requiere una solicitud única en la base de datos para que se devuelvan todos los objetos definidos por la ruta. El uso de rutas de consulta puede dar lugar a la ejecución de comandos complejos en el origen de datos partiendo de consultas de objeto aparentemente simples. Esto se produce porque se requieren una o más combinaciones para la devolución de objetos relacionados en una consulta única. Esta complejidad es mayor en consultas con un modelo de entidades complejo, como una entidad con herencia o una ruta que incluye relaciones varios a varios.

> **NOTE**
>
> Use el método ToTraceString para ver el comando que se generará mediante una consulta ObjectQuery<T>. Para obtener más información, consulte Cómo: ver los comandos de la tienda.

Cuando la ruta de una consulta incluye demasiados objetos relacionados o cuando los objetos contienen demasiados datos de fila, es posible que el origen de datos no pueda completar la consulta. Esto sucede si la consulta requiere un almacenamiento temporal intermedio que supera la capacidad del origen de datos. Cuando esto se produce, se puede reducir la complejidad de la consulta del origen de datos cargando explícitamente los objetos relacionados.

Cargar explícitamente los objetos relacionados

Puede cargar explícitamente los objetos relacionados llamando al método `Load` de una propiedad de navegación que devuelve una EntityCollection<TEntity> o EntityReference<TEntity>. La carga explícita de objetos requiere un viaje de ida y vuelta a la base de datos cada vez que se llama al método `Load`.

> **NOTE**
>
> Si llama al método `Load` mientras se recorre en bucle una colección de objetos devueltos, como cuando se usa la instrucción `foreach` (`For Each` en Visual Basic), el proveedor específico del origen de datos debe admitir varios conjuntos de resultados activos en una sola conexión. En una base de datos de SQL Server, debe especificar un valor `MultipleActiveResultSets = true` en la cadena de conexión del proveedor.

También puede utilizar el método LoadProperty cuando no haya propiedades EntityCollection<TEntity> o EntityReference<TEntity> en entidades. Esto es útil cuando se usan entidades POCO.

Aunque la carga explícita de objetos relacionados reducirá el número de combinaciones y la cantidad de datos redundantes, `Load` requiere varias conexiones a la base de datos, lo que puede resultar costoso cuando se carga explícitamente un gran número de objetos.

Guardar los cambios

Cuando se llama al método SaveChanges de una instancia de ObjectContext, se genera un comando de creación, actualización o eliminación independiente para cada objeto agregado, actualizado o eliminado en el contexto. Estos comandos se ejecutan en el origen de datos en una transacción única. Al igual que ocurre en las consultas, el rendimiento de las operaciones de creación, actualización y eliminación depende de la complejidad de la asignación en el modelo conceptual.

Transacciones distribuidas

Las operaciones de una transacción explícita que requieren recursos administrados por el coordinador de transacciones distribuidas (DTC) serán mucho más caras que las operaciones similares que no requieren DTC. La promoción a DTC se producirá en las situaciones siguientes:

- Una transacción explícita con una operación en una base de datos de SQL Server 2000 u otro origen de datos que siempre promueve transacciones explícitas a DTC.
- Una transacción explícita con una operación en SQL Server 2005 cuando la Entity Framework administra la conexión. Esto se produce porque SQL Server 2005 promueve a DTC cada vez que se cierra y se vuelve a abrir una conexión dentro de una transacción única, que es el comportamiento predeterminado del Entity Framework. Esta promoción a DTC no se produce si se usa SQL Server 2008. Para evitar esta promoción al usar SQL Server 2005, se debe abrir y cerrar explícitamente la conexión dentro de la transacción. Para obtener más información, consulte Administración de conexiones y transacciones.

Las transacciones explícitas se usan cuando se ejecutan una o varias operaciones dentro de una transacción System.Transactions. Para obtener más información, consulte Administración de conexiones y transacciones.

Estrategias para mejorar el rendimiento

Para mejorar el rendimiento global de las consultas en Entity Framework, use las estrategias siguientes.

Generar previamente las vistas

La generación de vistas basadas en un modelo de entidades supone un costo significativo la primera vez que una aplicación ejecuta una consulta. Emplee la utilidad EdmGen.exe para generar previamente las vistas en forma de archivo de código de Visual Basic o de C# que se puede agregar al proyecto durante el diseño. También podría utilizar el Kit de herramientas de transformación de plantillas de texto para generar vistas precompiladas. Las vistas generadas previamente se validan en tiempo de ejecución para garantizar su coherencia con la versión actual del modelo de entidades especificado. Para obtener más información, vea Cómo: generar previamente vistas para mejorar el rendimiento de las consultas.

Al trabajar con modelos muy grandes, se aplica la siguiente consideración:

El formato de metadatos de .NET limita a 16,777.215 (0xFFFFFF) el número de caracteres de cadena de usuario en un binario dado. Si va a generar vistas para un modelo muy grande y el archivo de vista alcanza este límite de tamaño, obtendrá el "no queda espacio lógico para crear más cadenas de usuario". error de compilación. Esta limitación de tamaño se aplica a todos los binarios administrados. Para obtener más información, consulte el blog que muestra cómo evitar el error al trabajar con modelos grandes y complejos.

Considerar la posibilidad de usar la opción de fusión mediante combinación NoTracking en las consultas

El seguimiento de los objetos devueltos en el contexto del objeto tiene un costo implícito. Para detectar cambios en los objetos y garantizar que varias solicitudes para la misma entidad lógica devuelvan la misma instancia del objeto, es necesario que los objetos se adjunten a una instancia de ObjectContext. Si no tiene previsto realizar actualizaciones o eliminaciones en los objetos y no requiere la administración de identidades, considere la posibilidad de usar las NoTracking Opciones de combinación al ejecutar las consultas.

Devolver la cantidad correcta de datos

En algunos escenarios, la especificación de una ruta de consulta mediante el método Include es mucho más rápida porque requiere menos recorridos de ida y vuelta a la base de datos. Sin embargo, en otros escenarios, los recorridos adicionales de ida y vuelta a la base de datos para cargar los objetos relacionados pueden ser más rápidos porque las consultas más sencillas con menos combinaciones producen una menor cantidad de datos redundantes. Por ello, se recomienda probar el rendimiento de varias maneras para recuperar los objetos relacionados. Para obtener más información, vea cargar objetos relacionados.

Para evitar devolver demasiados datos en una sola consulta, considere la posibilidad de paginar los resultados de la consulta en grupos que sean más fáciles de administrar. Para obtener más información, vea Cómo: paginar a

través de los resultados de la consulta.

Limitar el ámbito de ObjectContext

En la mayoría de los casos, deberá crear una instancia de ObjectContext dentro de una instrucción `using` (`Using…End Using` en Visual Basic). Esto puede aumentar el rendimiento garantizando que los recursos asociados al contexto del objeto se eliminan automáticamente cuando el código sale del bloque de instrucciones. Sin embargo, cuando los controles se enlazan a objetos administrados por el contexto del objeto, se debe mantener la instancia de ObjectContext mientras se necesite el enlace, y eliminarla posteriormente de forma manual. Para obtener más información, consulte Administración de conexiones y transacciones.

Considerar la posibilidad de abrir manualmente la conexión de la base de datos

Cuando la aplicación ejecuta una serie de consultas de objeto o llamadas con frecuencia SaveChanges para conservar las operaciones de creación, actualización y eliminación en el origen de datos, el Entity Framework debe abrir y cerrar continuamente la conexión al origen de datos. En estos casos, considere la posibilidad de abrir manualmente la conexión al comienzo de estas operaciones y cerrarla o eliminarla una vez finalizadas dichas operaciones. Para obtener más información, consulte Administración de conexiones y transacciones.

Datos de rendimiento

Algunos datos de rendimiento del Entity Framework se publican en las siguientes publicaciones en el blog del equipo de ADO.net:

- Explorar el rendimiento de ADO.NET Entity Framework - Parte 1
- Explorar el rendimiento de ADO.NET Entity Framework – Parte 2
- Comparación del rendimiento de ADO.NET Entity Framework

Consulte también

- Consideraciones de desarrollo e implementación

Consideraciones de migración (Entity Framework)

30/11/2019 • 16 minutes to read • Edit Online

El Entity Framework ADO.NET proporciona varias ventajas a una aplicación existente. Una de las más importantes es la capacidad de utilizar el modelo conceptual para separar las estructuras de datos que usa la aplicación del esquema en el origen de datos. De esta forma se pueden realizar más adelante y con facilidad cambios en el modelo de almacenamiento o en el propio origen de datos sin realizar los cambios correspondientes en la aplicación. Para obtener más información acerca de las ventajas de usar el Entity Framework, consulte Entity Framework información general y Entity Data Model.

Para aprovechar las ventajas de los Entity Framework, puede migrar una aplicación existente a la Entity Framework. Algunas tareas son comunes a todas las aplicaciones migradas. Estas tareas comunes incluyen la actualización de la aplicación para usar el .NET Framework a partir de la versión 3,5 Service Pack 1 (SP1), la definición de modelos y la asignación y la configuración del Entity Framework. Al migrar una aplicación al Entity Framework, existen consideraciones adicionales que se aplican. Estas consideraciones dependen del tipo de aplicación que se migra y de la funcionalidad concreta de la aplicación. Este tema proporciona información para ayudarle a elegir el mejor enfoque que utilizar al actualizar una aplicación existente.

Consideraciones generales de la migración

Las consideraciones siguientes se aplican al migrar cualquier aplicación a la Entity Framework:

- Cualquier aplicación que use el .NET Framework a partir de la versión 3,5 SP1 se puede migrar al Entity Framework, siempre que el proveedor de datos para el origen de datos utilizado por la aplicación admita la Entity Framework.
- Entity Framework puede no admitir toda la funcionalidad de un proveedor de origen de datos, aun cuando ese proveedor sí admita Entity Framework.
- En una aplicación grande o compleja, no es necesario migrar toda la aplicación a Entity Framework a la vez. Sin embargo, cualquier parte de la aplicación que no use Entity Framework aún debe cambiarse cuando el origen de datos cambie.
- La conexión del proveedor de datos utilizada por el Entity Framework se puede compartir con otras partes de la aplicación porque el Entity Framework utiliza proveedores de datos ADO.NET para tener acceso al origen de datos. Por ejemplo, Entity Framework utiliza el proveedor SqlClient para tener acceso a una base de datos de SQL Server. Para obtener más información, consulte proveedor de EntityClient para el Entity Framework.

Tareas de migración comunes

La ruta de acceso para migrar una aplicación existente al Entity Framework depende tanto del tipo de aplicación como de la estrategia de acceso a datos existente. Sin embargo, siempre debe realizar las siguientes tareas al migrar una aplicación existente al Entity Framework.

> **NOTE**
> Todas estas tareas se realizan automáticamente al usar las herramientas de Entity Data Model a partir de Visual Studio 2008. Para obtener más información, consulte Cómo: usar el Asistente para Entity Data Model.

1. Actualice la aplicación.

Un proyecto creado con una versión anterior de Visual Studio y el .NET Framework debe actualizarse para usar Visual Studio 2008 SP1 y el .NET Framework a partir de la versión 3,5 SP1.

2. Defina los modelos y la asignación.

   Los archivos de modelo y asignación definen las entidades en el modelo conceptual; las estructuras en el origen de datos, por ejemplo las tablas, los procedimientos almacenados y vistas; y la asignación entre las entidades y estructuras del origen de datos. Para obtener más información, vea Cómo: definir manualmente los archivos de asignación y de modelo.

   Los tipos que se definen en el modelo de almacenamiento deben coincidir con el nombre de los objetos en el origen de datos. Si la aplicación existente expone los datos como objetos, debe asegurarse de que las entidades y las propiedades que se definen en el modelo conceptual coincidan con los nombres de estas clases de datos y propiedades existentes. Para obtener más información, vea Cómo: personalizar el modelado y la asignación de archivos para trabajar con objetos personalizados.

   > **NOTE**
   > Entity Data Model Designer se puede utilizar para cambiar el nombre de las entidades en el modelo conceptual de modo que coincidan con los objetos existentes. Para obtener más información, vea Diseñador de Entity Data Model.

3. Defina la cadena de conexión.

   El Entity Framework usa una cadena de conexión con formato especial al ejecutar las consultas en un modelo conceptual. Esta cadena de conexión encapsula la información sobre los archivos de modelo y asignación y la conexión al origen de datos. Para obtener más información, consulte Cómo: definir la cadena de conexión.

4. Configure el proyecto de Visual Studio.

   Las referencias a los ensamblados de Entity Framework y los archivos de asignación y de modelo se deben agregar al proyecto de Visual Studio. Puede agregar estos archivos de asignación al proyecto para asegurarse de que se implementan con la aplicación en la ubicación que se indica en la cadena de conexión. Para obtener más información, consulte Cómo: configurar manualmente un proyecto de Entity Framework.

Consideraciones para las aplicaciones con objetos existentes

A partir de la .NET Framework 4, el Entity Framework admite objetos CLR "antiguos sin formato" (POCO), también denominados objetos que ignoran la persistencia. En la mayoría de los casos, los objetos existentes pueden trabajar con el Entity Framework realizando cambios menores. Para obtener más información, vea trabajar con entidades poco. También puede migrar una aplicación a la Entity Framework y usar las clases de datos generadas por las herramientas de Entity Framework. Para obtener más información, consulte Cómo: usar el Asistente para Entity Data Model.

Consideraciones para las aplicaciones que utilizan los proveedores ADO.NET

Los proveedores de ADO.NET, como SqlClient, permiten consultar un origen de datos para devolver datos tabulares. Los datos también se pueden cargar en un conjunto de datos de ADO.NET. En la lista siguiente se describen las consideraciones para actualizar una aplicación que usa un proveedor de ADO.NET existente:

- Muestre los datos tabulares utilizando un lector de datos.

  Puede considerar la ejecución de una consulta de Entity SQL mediante el proveedor de EntityClient y la enumeración a través del objeto de EntityDataReader devuelto. Haga esto solo si la aplicación muestra datos

tabulares mediante un lector de datos y no requiere los medios proporcionados por el Entity Framework para materializar los datos en los objetos, realizar el seguimiento de los cambios y realizar actualizaciones. Puede continuar utilizando el código de acceso a los datos existente que realiza las actualizaciones en el origen de datos, pero puede utilizar la conexión existente a la que se obtiene acceso desde la propiedad StoreConnection de EntityConnection. Para obtener más información, consulte proveedor de EntityClient para el Entity Framework.

- Trabaje con objetos DataSet.

  El Entity Framework proporciona muchas de las mismas funcionalidades proporcionadas por el conjunto de datos, incluida la persistencia en memoria, el seguimiento de los cambios, el enlace de datos y la serialización de objetos como datos XML. Para obtener más información, vea trabajar con objetos.

  Si el Entity Framework no proporciona la funcionalidad del conjunto de cambios que la aplicación necesita, puede aprovechar las ventajas de las consultas LINQ mediante LINQ to DataSet. Para más información, vea LINQ to DataSet.

Consideraciones para las aplicaciones que enlazan datos a los controles

El .NET Framework permite encapsular los datos en un origen de datos, como un conjunto de datos o un control de origen de datos ASP.NET y, a continuación, enlazar los elementos de la interfaz de usuario a esos controles de datos. La lista siguiente describe las consideraciones del enlace de los controles a los datos de Entity Framework.

- Enlace los datos a los controles.

  Al consultar el modelo conceptual, el Entity Framework devuelve los datos como objetos que son instancias de tipos de entidad. Estos objetos se pueden enlazar directamente a los controles y este enlace admite actualizaciones. Esto significa que los cambios en los datos de un control, como una fila de un DataGridView, se guardan automáticamente en la base de datos cuando se llama al método SaveChanges.

  Si una aplicación enumera los resultados de una consulta para mostrar los datos en un control de tipo DataGridView o de otro tipo que admite el enlace de datos, puede modificar la aplicación para enlazar el control al resultado de ObjectQuery<T>.

  Para obtener más información, vea enlazar objetos a controles.

- Controles de origen de datos ASP.NET.

  El Entity Framework incluye un control de origen de datos diseñado para simplificar el enlace de datos en las aplicaciones Web de ASP.NET. Para obtener más información, vea información general sobre el control de servidor Web EntityDataSource.

Otras consideraciones

Las siguientes son consideraciones que se pueden aplicar al migrar tipos específicos de aplicaciones a Entity Framework.

- Aplicaciones que exponen los servicios de los datos.

  Los servicios Web y las aplicaciones que se basan en Windows Communication Foundation (WCF) exponen los datos de un origen de datos subyacente utilizando un formato de mensajería de solicitud/respuesta XML. El Entity Framework admite la serialización de objetos entidad mediante la serialización binaria, XML o de contrato de datos de WCF. Las serializaciones WCF y binaria admiten ambas la serialización completa de los gráficos de objetos. Para obtener más información, vea compilar aplicaciones de N niveles.

- Aplicaciones que utilizan datos XML.

  La serialización de objetos permite crear Entity Framework Data Services. Estos servicios proporcionan

datos a las aplicaciones que consumen datos XML, como las aplicaciones de Internet basadas en AJAX. En estos casos, considere la posibilidad de usar WCF Data Services. Estos servicios de datos se basan en el Entity Data Model y proporcionan acceso dinámico a los datos de la entidad mediante acciones HTTP de transferencia de estado de representación (REST) estándar, como GET, PUT y POST. Para obtener más información, vea WCF Data Services 4.5.

El Entity Framework no admite un tipo de datos XML nativo. Esto significa que cuando una entidad se asigna a una tabla con una columna XML, la propiedad de entidad equivalente para la columna XML es una cadena. Los objetos se pueden desconectar y serializar como XML. Para obtener más información, vea serializar objetos.

Si una aplicación requiere la capacidad de consultar datos XML, todavía puede aprovecharse de las ventajas de las consultas LINQ utilizando LINQ to XML. Para obtener más información, vea LINQ to XMLC#() o LINQ to XML (Visual Basic).

- Aplicaciones que mantienen el estado.

  Las aplicaciones Web de ASP.NET deben mantener con frecuencia el estado de una página web o de una sesión de usuario. Los objetos de una instancia de ObjectContext pueden almacenarse en el estado de vista de cliente o en el estado de sesión en el servidor y, posteriormente, recuperarse y volver a adjuntarse a un nuevo contexto de objeto. Para obtener más información, vea adjuntar y separar objetos.

Vea también

- Consideraciones de implementación
- Terminología de Entity Framework

Consideraciones de implementación (Entity Framework)

23/10/2019 • 3 minutes to read • Edit Online

En este tema se proporciona información sobre cómo implementar aplicaciones que utilizan ADO.NET Entity Framework para el acceso a datos. Para obtener más información sobre el Entity Framework, vea Introducción.

Entity Framework proporciona un conjunto de herramientas que se integran con Visual Studio y facilitan el desarrollo. Para obtener más información, consulte ADO.NET Entity Data Model Tools. En este tema no se describe cómo utilizar tecnologías concretas para implementar una aplicación basada en Entity Framework.

Visual Studio proporciona funcionalidad para distribuir e implementar aplicaciones, como la implementación ClickOnce. Para obtener más información, vea implementar aplicaciones y componentes en la documentación de Visual Studio.

Las consideraciones siguientes se aplican al implementar una aplicación que utiliza Entity Framework:

- Entity Framework forma parte de .NET Framework a partir de .NET Framework 3.5 Service Pack 1 (SP1). Debe asegurarse de que esté instalado .NET Framework 3.5 Service Pack 1 o posterior al implementar una aplicación basada en Entity Framework.
- Cuando el Asistente para Entity Data Model genera un modelo conceptual, se crea una cadena de conexión en el archivo de configuración de la aplicación. Los archivos de asignación y de modelo se pueden incrustar como recursos de aplicación o se pueden copiar en el directorio de resultados. De forma predeterminada, se implementan como recursos incrustados de la aplicación. Use la propiedad `Metadata Artifact Processing` del archivo de Entity Designer para seleccionar una de estas opciones. Para obtener más información, consulte Cómo Copie los archivos de asignación y de modelo enel directorio de salida.
- Asegúrese de que la información sobre la asignación y el modelo (expresada en el lenguaje de definición de esquemas conceptuales (CSDL), el lenguaje de definición de esquemas de almacenamiento (SSDL) y el lenguaje de especificación de asignaciones (MSL)) se implementa junto con la aplicación y en la ubicación especificada por la cadena de conexión. Para más información, consulte Cadenas de conexión.
- Cuando se incrusta información sobre la asignación y el modelo como recursos de la aplicación, se debe recompilar e implementar la aplicación cada vez que se actualiza el modelo conceptual.
- Dado que Entity Framework es un componente de .NET Framework, se puede redistribuir con la aplicación si lo permite el contrato de licencia de .NET Framework.

Vea también

- ADO.NET Entity Framework
- Consideraciones de desarrollo e implementación

Recursos de Entity Framework

20/02/2020 • 2 minutes to read • Edit Online

Los recursos externos siguientes proporcionan información y compatibilidad para la creación de aplicaciones de Entity Framework.

Blog del equipo de ADO.NET
Blog que contiene novedades y análisis de las características y funciones de ADO.NET.

Blog de diseño de Entity Framework
Debates sobre el diseño y presentaciones de característica para futuras versiones de Entity Framework.

Preguntas más frecuentes sobre Entity Framework
Sección de wiki que contiene las preguntas más frecuentes sobre el Entity Framework.

Consulte también

- Información general sobre Entity Framework
- Introducción
- Terminología de Entity Framework
- ADO.NET Entity Data Model herramientas

Terminología de Entity Framework

24/04/2020 • 21 minutes to read • Edit Online

En este tema se definen los términos a los que se hace referencia con frecuencia en la documentación de Entity Framework. Se proporcionan vínculos a temas importantes donde hay información adicional.

TÉRMINO	DEFINICIÓN
asociación	Definición de una relación entre tipos de entidad. Para obtener más información, vea Elemento de asociación (CSDL) y tipo de asociación.
conjunto de asociaciones	Contenedor lógico para instancias de asociaciones del mismo tipo. Para obtener más información, vea Elemento AssociationSet (CSDL) y conjunto de asociaciones.
Code First	A partir de Entity Framework 4.0.1 puede crear un modelo mediante programación usando desarrollo Code First. Hay dos escenarios diferentes para el desarrollo Code First. En ambos casos, el desarrollador define un modelo codificando definiciones de clase de .NET Framework y especifica opcionalmente la asignación o configuración adicional usando anotaciones de datos o la API fluida. El desarrollo Code First forma parte de Entity Framework 5.0. Entity Framework 5.0 no forma parte de .NET Framework, pero se basa en .NET Framework 4.5. Entity Framework 5.0 está disponible como el paquete NuGet de Entity Framework. Para obtener más información, vea Versiones anteriores de Entity Framework.
árbol de comandos	Representación mediante programación común de todas las consultas de Entity Framework que se componen de una o varias expresiones. Para obtener más información, vea Información general de Entity Framework.
tipo complejo	Clase de .NET Framework que representa una propiedad compleja tal y como se define en el modelo conceptual. Los tipos complejos permiten que las propiedades escalares se organicen dentro de entidades. Los objetos complejos son instancias de los tipos complejos. Para obtener más información, vea ComplexType Element (CSDL) y complex type.
ComplexType	Especificación de un tipo de datos que representa una propiedad no escalar de un tipo de entidad que no tiene una propiedad clave. Para obtener más información, vea ComplexType Element (CSDL) y complex type.

TÉRMINO	DEFINICIÓN
modelo conceptual	Especificación abstracta para los tipos de entidad, tipos complejos, asociaciones, contenedores de entidades, conjuntos de entidades y conjuntos de asociaciones en el dominio de una aplicación en Entity Framework. El modelo conceptual se define en CSDL en el archivo .csdl. Para obtener más información, consulte Modelado y asignación.
archivo .csdl	Archivo XML que contiene el modelo conceptual, expresado en CSDL.
lenguaje de definición de esquemas conceptual (CSDL)	Lenguaje basado en XML que se utiliza para definir tipos de entidades, asociaciones, contenedores de entidades, conjuntos de entidades y conjuntos de asociaciones de un modelo conceptual. Para obtener más información, consulta CSDL Specification.
contenedor	Agrupación lógica de conjuntos de entidades y de asociaciones. Para obtener más información, vea EntityContainer Element (CSDL) y entity container.
simultaneidad	Proceso que permite a varios usuarios tener acceso a, y modificar, datos compartidos al mismo tiempo. De forma predeterminada, Entity Framework implementa un modelo de simultaneidad optimista.
direction	Hace referencia a la naturaleza asimétrica de algunas asociaciones. La dirección se especifica con los atributos `FromRole` y `ToRole` de un elemento `NavigationProperty` o `ReferentialConstraint` en un esquema. Para obtener más información, vea Elemento NavigationProperty (CSDL) y propiedad de navegación.
carga diligente	El proceso de carga de un conjunto específico de objetos relacionados junto con los objetos que se solicitaron explícitamente en la consulta.
.edmx (archivo)	Archivo XML que contiene el modelo conceptual (en CSDL), el modelo de almacenamiento (en SSDL) y las asignaciones entre ellos (en MSL). Entity Data Model Tools crea el archivo .edmx. Para obtener más información, consulte Información general sobre archivos .edmx.
end	Entidad participante en una asociación. Para obtener más información, vea End Element (CSDL) y association end.

TÉRMINO	DEFINICIÓN
Entidad	Concepto en el dominio de una aplicación partir del que se define un tipo de datos. Para obtener más información, vea EntityType Element (CSDL) y tipo de entidad.
EntityClient	Un proveedor de datos de ADO.NET `EntityConnection` independiente `EntityCommand` del `EntityDataReader` almacenamiento que contiene clases como , , y . Funciona Entity SQL con y se conecta a proveedores de datos de ADO.NET de almacenamiento específicos, como `SqlClient` . Para obtener más información, consulte Proveedor de EntityClient para Entity Framework.
contenedor de entidades	Especifica los conjuntos de entidades y los conjuntos de asociaciones que se implementarán en un espacio de nombres especificado. Para obtener más información, vea EntityContainer Element (CSDL) y entity container.
Entity Data Model (EDM)	Conjunto de conceptos que describen la estructura de los datos, como entidades y relaciones, independientemente del formato en el que estén almacenados. Para obtener más información, vea Entity Data Model.
Entity Framework	Conjunto de tecnologías que admite el desarrollo de aplicaciones de software orientadas a datos permitiendo a los programadores trabajar con modelos conceptuales que se asignan a los esquemas lógicos en los orígenes de datos. Para obtener más información, vea Información general de Entity Framework.
conjunto de entidades	Contenedor lógico de entidades de un tipo determinado y sus subtipos. Los conjuntos de entidades se asignan a las tablas en una base de datos. Para obtener más información, vea EntitySet Element (CSDL) y entity set.
Entity SQL	Dialecto de SQL, independiente del almacenamiento, que trabaja directamente con esquemas de entidades conceptuales y admite características de modelos conceptuales, como la herencia y las relaciones. Para obtener más información, vea Lenguaje Entity SQL.
tipo de entidad	Clase de .NET Framework que representa una entidad tal como se define en el modelo conceptual. Los tipos de entidad pueden tener propiedades de navegación, escalares y complejas. Los objetos son instancias de tipos de entidad. Para obtener más información, consulte Trabajar con objetos.

TÉRMINO	DEFINICIÓN
EntityType	Especificación de un tipo de datos que incluye una clave y un conjunto con nombre de propiedades y representa un elemento de nivel superior en un modelo conceptual o modelo de almacenamiento. Para obtener más información, vea EntityType Element (CSDL) y tipo de entidad.
carga explícita	Cuando una consulta devuelve objetos, los objetos relacionados no se cargan al mismo tiempo. De forma predeterminada, los objetos no se cargan hasta que se solicita explícitamente utilizando el método `Load` sobre una propiedad de navegación.
asociación de clave externa	Asociación entre entidades que se administra a través de las propiedades de una clave externa.
relación de identificación	Relación donde la clave principal de la entidad principal también forma parte de la clave principal de la entidad dependiente. En este tipo de relación, la entidad dependiente no puede existir sin la entidad principal.
asociación independiente	Asociación entre entidades que se representa y es objeto de seguimiento mediante un objeto independiente.
key	Atributo de un tipo de entidad que especifica qué propiedad o conjunto de propiedades se utiliza para identificar instancias únicas del tipo de entidad. Se representa en el nivel de objetos mediante la clase EntityKey. Para obtener más información, vea Elemento clave (CSDL) y clave de entidad.
carga diferida	Cuando una consulta devuelve objetos, los objetos relacionados no se cargan al mismo tiempo. En vez de ello, se cargan automáticamente cuando se obtiene acceso a la propiedad de navegación.
LINQ to Entities	Sintaxis de consulta que define un conjunto de operadores de consulta que permiten que las operaciones de recorrido, filtro y proyección se expresen de forma directa y declarativa en Visual C. Para obtener más información, vea LINQ to Entities.
mapping	Especificación de las correspondencias entre los elementos de un modelo conceptual y los elementos de un modelo de almacenamiento. Para obtener más información, consulte Especificación de MSL.
archivo .msl	Archivo XML que contiene la asignación entre el modelo conceptual y el modelo de almacenamiento, expresado en MSL.

TÉRMINO	DEFINICIÓN
lenguaje de especificación de asignaciones (MLS)	Lenguaje basado en XML que se utiliza para asignar los elementos definidos en un modelo conceptual a los elementos de un modelo de almacenamiento. Para obtener más información, consulte Especificación de MSL.
funciones de modificación	Procedimientos almacenados que se utilizan para insertar, actualizar y eliminar los datos que están en el origen de datos. Estas funciones se utilizan en lugar de comandos generados por Entity Framework. El elemento `Function` define las funciones de modificación en el modelo de almacenamiento. El elemento ModificationFunctionMapping asigna estas funciones de modificación a las operaciones de inserción, actualización y eliminación en entidades definidas en el modelo conceptual.
multiplicidad	Número de entidades que pueden existir en cada lado de una relación, tal y como define una asociación. Se conoce también como cardinalidad. Para obtener más información, vea End Element (CSDL) y association end.
multiple entity sets per type	Capacidad de definir un tipo de entidad en más de un conjunto de entidades. Para obtener más información, vea EntitySet Element (CSDL) y How to: Define a Model with Multiple Entity Sets per Type.
propiedad de navegación	Propiedad de un tipo de entidad que representa una relación con otro tipo de entidad, tal y como se define mediante una asociación. Las propiedades de navegación se utilizan para devolver los objetos relacionados como EntityCollection<TEntity> o EntityReference<TEntity>, dependiendo de la multiplicidad en el otro extremo de la asociación. Para obtener más información, vea Elemento NavigationProperty (CSDL) y propiedad de navegación.
ruta de consulta	Representación de cadena de una ruta de acceso que especifica qué objetos relacionados devolver cuando se ejecuta una consulta de objeto. Una ruta de consulta se define llamando al método Include en ObjectQuery<T>. Para obtener más información, consulte Carga de objetos relacionados.

TÉRMINO	DEFINICIÓN
contexto de objeto	Representa el contenedor de la entidad definido en el modelo conceptual. Contiene una conexión al origen de datos subyacente y proporciona servicios como el seguimiento de cambios y la resolución de identidad. Una instancia de la clase ObjectContext o `DbContext` representa un contexto de objeto. `DbContext` forma parte de Entity Framework 5.0. Entity Framework 5.0 no forma parte de .NET Framework, pero se basa en .NET Framework 4.5. Entity Framework 5.0 está disponible como el paquete NuGet de Entity Framework. Para obtener más información, vea Versiones anteriores de Entity Framework.
nivel de objeto	Los tipos de entidad y las definiciones de contexto del objeto que utiliza Entity Framework.
consulta de objeto	Consulta ejecutada contra un modelo conceptual, dentro del contexto de un objeto, que devuelve los datos en forma de objetos. Para obtener más información, consulte Consultas de objetos.
asignación objeto relacional	Técnica para transformar los datos de una base de datos relacional en tipos de datos que se pueden utilizar en aplicaciones de software orientadas a objetos. Entity Framework proporciona servicios de asignación relacional de objetos mediante la asignación de datos relacionales, tal como se define en el modelo de almacenamiento, a tipos de datos, tal como se define en el modelo conceptual. Para obtener más información, consulte Modelado y asignación.
Servicios de objeto	Servicios proporcionados por Entity Framework que permiten que el código de aplicación funcione en entidades como objetos de .NET Framework.
objeto que ignora la persistencia	Objeto que no contiene ninguna lógica relacionada con el almacenamiento de datos. Se conoce también como entidad POCO.
POCO	Plain Old CLR Object (objetos CLR antiguos sin formato). Objeto que no hereda de otra clase o implementa una interfaz.
entidad POCO	Una entidad de Entity Framework que EntityObject ComplexObject no hereda de las interfaces de Entity Framework o no implementa las interfaces. Con frecuencia, las entidades POCO son objetos de dominio existentes que se usan en una aplicación de Entity Framework. Estas entidades pueden ignorar la persistencia. Para obtener más información, consulte Trabajar con entidades POCO.

TÉRMINO	DEFINICIÓN
objeto proxy	Objeto que deriva de una clase POCO y que Entity Framework genera para admitir el seguimiento de cambios y la carga diferida. Para obtener más información, consulte Requisitos para crear servidores proxy POCO.
restricción referencial	Restricción que se define en un modelo conceptual y que indica que una entidad tiene una relación de dependencia con otra entidad. Esta restricción significa que una instancia de una entidad dependiente no puede existir sin una instancia correspondiente de la entidad de seguridad. Para obtener más información, vea Elemento ReferentialConstraint (CSDL) y Restricción de integridad referencial.
relación	Conexión lógica entre entidades.
rol	Nombre dado a cada `End` de una asociación para clarificar la semántica de la relación. Para obtener más información, vea End Element (CSDL) y association end.
propiedad escalar	Propiedad de una entidad que se asigna a un único campo en el modelo de almacenamiento.
entidad de autoseguimiento	Una entidad creada desde una plantilla del Kit de herramientas de transformación de plantillas de texto (T4) con la capacidad de registrar cambios en las propiedades escalares, complejas y de navegación.
ipo simple	Tipo primitivo que se utiliza para definir las propiedades en el modelo conceptual. Para obtener más información, vea Tipos de modelo conceptual (CSDL) y Entity Data Model: Tipos de datos primitivos.
entidad dividida	Tipo de entidad que está asignada a dos tipos independientes en el modelo de almacenamiento. Para obtener más información, vea Cómo: definir un modelo con una entidad única asignada a dos tablas.
modelo de almacenamiento	Definición del modelo lógico de datos en un origen de datos admitido, como una base de datos relacional. El modelo de almacenamiento se define en SSDL en el archivo .ssdl. Para obtener más información, consulte Modelado y asignación y Especificación SSDL.
archivo .ssdl	Archivo XML que contiene el modelo de almacenamiento, expresado en SSDL.

TÉRMINO	DEFINICIÓN
lenguaje de definición de esquemas de almacenamiento (SSDL)	Lenguaje basado en XML que se utiliza para definir los tipos de entidad, asociaciones, contenedores de entidad, conjuntos de entidades y conjuntos de asociaciones de un modelo de almacenamiento que, con frecuencia, corresponden a un esquema de base de datos. Para obtener más información, consulte Especificación SSDL.
tabla por jerarquía	Método de modelado de una jerarquía de tipos de una base de datos que incluye los atributos de todos los tipos de la jerarquía en una tabla.
tabla por tipo	Método de modelado de una jerarquía de tipos de una base de datos que usa varias tablas con relaciones uno a uno para modelar los distintos tipos.

Consulte también

- ADO.NET Entity Framework
- Información general sobre Entity Framework
- Introducción
- Recursos de Entity Framework

Introducción (Entity Framework)

20/02/2020 • 2 minutes to read • Edit Online

ADO.NET Entity Framework admite Aplicaciones y servicios centrados en datos, y proporciona una plataforma para programar con datos que eleva el nivel de abstracción del nivel lógico relacional al nivel conceptual. Al permitir a los desarrolladores trabajar con datos en un nivel de abstracción mayor, el Entity Framework admite código que es independiente de cualquier motor de almacenamiento de datos o esquema relacional determinados. Para obtener más información, vea información general sobre Entity Framework.

Para empezar a usar rápidamente la versión más reciente de la Entity Framework, consulte Introducción a Entity Framework 6.

Consulte también

- Entity Framework

Referencia del lenguaje Entity SQL

11/01/2020 • 2 minutes to read • Edit Online

En esta sección se proporciona documentación detallada LINQ to Entities, Entity SQL y los lenguajes de modelado y asignación utilizados por el Entity Framework.

En esta sección

- LINQ to Entities
- Lenguaje Entity SQL
- Funciones canónicas

Secciones relacionadas

- ADO.NET Entity Data Model herramientas

Vea también

- ADO.NET Entity Framework
- Introducción
- Ejemplos
- Especificaciones CSDL, SSDL MSL