UNIVERSIDAD TECNOLÓGICA DEL PERÚ

LINQ TO ENTITIES

I OBJETIVOS

- Descubrir la sintaxis de LINQ to Entities.
- Conocer el manejo de LINQ to Entities.

II TEMAS A TRATAR

- Introducción a consultas con LINQ to Entities

III MARCO TEORICO

LINQ TO ENTITIES

LINQ to Entities proporciona la capacidad de realizar consultas integradas en lenguajes (LINQ) que permite a los desarrolladores de software escribir consultas contra el modelo conceptual de Entity Framework mediante Visual Basic o Visual C#. Las consultas con Entity Framework se representan mediante consultas de árboles de comandos, que se ejecutan en el contexto del objeto.

LINQ to Entities convierte las consultas de Language-Integrated Queries (LINQ) en consultas de árboles de comandos, ejecuta las consultas en Entity Framework y devuelve objetos que se pueden usar tanto en Entity Framework como en LINQ. A continuación se muestra el proceso para crear y ejecutar una consulta de LINQ to Entities.

1. Cree una instancia de ObjectQuery<T> en ObjectContext.
2. Cree una consulta de LINQ to Entities en C# o Visual Basic con la instancia de ObjectQuery<T>.
3. Convierta los operadores y expresiones de consulta estándar de LINQ en árboles de comandos.
4. Ejecute la consulta, con representación de un árbol de comandos, en el origen de datos. Las excepciones producidas en el origen de datos durante la ejecución se pasan directamente al cliente.

5. Devuelva los resultados de la consulta al cliente.

CREAR UNA INSTANCIA DE OBJECTQUERY

La clase ObjectQuery<T> genérica representa una consulta que devuelve una colección de cero o más **entidades con tipo**. Una consulta de objetos se suele crear a partir de un contexto del objeto existente, en lugar de crearse de forma manual, y siempre pertenece a dicho contexto. Este contexto proporciona la información de metadatos y de la conexión necesaria para crear y ejecutar la consulta.

La clase **genérica ObjectQuery<T>** implementa la interfaz **genérica IQueryable<T>**, cuyos métodos de generador permiten que las consultas de LINQ se generen de forma gradual. También puede dejar que el compilador deduzca el tipo de entidades utilizando la palabra clave **var** de C# (**Dim** en Visual Basic, con la inferencia de tipos locales habilitada).

CREAR CONSULTAS

Las instancias de la clase genérica ObjectQuery<T>, que implementa la interfaz genérica IQueryable<T>, actúan como origen de datos para las consultas de LINQ to Entities. En una consulta se especifica exactamente la información que se desea recuperar del origen de datos. Una consulta también puede especificar cómo se debe ordenar, agrupar y conformar esa información antes de que se devuelva. En LINQ, una consulta se almacena en una variable. Esta variable de consulta no realiza ninguna acción y no devuelve datos; solamente almacena la información de la consulta. Tras crear una consulta debe ejecutarla para recuperar los datos.

Las consultas de LINQ to Entities se pueden formular en dos sintaxis diferentes: **sintaxis de expresiones de consulta y sintaxis de consultas basadas en métodos**.

Sintaxis de expresiones de consulta

Las expresiones de consulta son una sintaxis de consulta declarativa. Esta sintaxis permite a un programador escribir consultas en un lenguaje de alto nivel que tenga un formato similar al de Transact-SQL. Si se utiliza la sintaxis de expresiones de consulta, se pueden realizar incluso operaciones complejas de filtrado, ordenación y agrupamiento en orígenes de datos con código mínimo.

En el ejemplo siguiente se usa Select para devolver todas las filas de *Product* y mostrar los nombres de producto.

```
using (AdventureWorksEntities AWEntities = new AdventureWorksEntities())
{
    ObjectQuery<Product> products = AWEntities.Product;

    IQueryable<string> productNames =
       from p in products
       select p.Name;

    Console.WriteLine("Product Names:");
```

```
    foreach (var productName in productNames)
    {
        Console.WriteLine(productName);
    }
}
```

Sintaxis de consultas basadas en métodos

Otra forma de formular consultas de LINQ to Entities es usar las consultas basadas en métodos. La sintaxis de consultas basadas en métodos es una secuencia de llamadas directas a los métodos de operador de LINQ que pasan expresiones lambda como parámetros.

En este ejemplo se usa **Select** para devolver todas las filas de *Product* y mostrar los nombres de producto.

```
using (AdventureWorksEntities AWEntities = new AdventureWorksEntities())
{
    ObjectQuery<Product> products = AWEntities.Product;

    IQueryable<string> productNames = products.Select(p => p.Name);


    Console.WriteLine("Product Names:");
    foreach (var productName in productNames)
    {
        Console.WriteLine(productName);
    }
}
```

CONSULTAS

Tal y como se ha mencionado anteriormente en este tema, la variable de consulta sólo almacena los comandos de la consulta cuando ésta se diseña para devolver una secuencia de valores. Si la consulta no contiene un método que origine una ejecución inmediata, la ejecución real de la consulta se aplazará hasta que la variable de consulta se recorra en iteración en un bucle **foreach** o **For Each**. La ejecución aplazada permite combinar varias consultas o ampliar una consulta. Cuando se amplía una consulta, se modifica para incluir las nuevas operaciones. La ejecución final reflejará los cambios.

En el siguiente ejemplo, la primera consulta devuelve todos los productos.

La segunda consulta amplía la primera usando **Where** para devolver todos los productos de tamaño "L":

```
using (AdventureWorksEntities AWEntities = new AdventureWorksEntities())
{

    ObjectQuery<Product> products = AWEntities.Product;

    IQueryable<Product> productsQuery =
        from p in products
        select p;

    IQueryable<Product> largeProducts = productsQuery.Where(p => p.Size == "L");
```

```
    Console.WriteLine("Products of size 'L':");
    foreach (var product in largeProducts)
    {
        Console.WriteLine(product.Name);
    }
}
```

Una vez ejecutada una consulta, todas las consultas sucesivas utilizarán los operadores de LINQ en memoria. Si se recorre en iteración la variable de la consulta utilizando una instrucción **foreach** o **For Each** o llamando a uno de los operadores de conversión de LINQ, se producirá la ejecución inmediata. Entre estos operadores de conversión se incluyen los siguientes: **ToList**, **ToArray**, ToLookup y ToDictionary.

En el siguiente ejemplo, la primera consulta devuelve todos los productos. La segunda consulta amplía la primera una vez que esta se haya ejecutado para devolver los productos de color rojo.

```
using (AdventureWorksEntities AWEntities = new AdventureWorksEntities())
{

    ObjectQuery<Product> products = AWEntities.Product;

    IQueryable<Product> productsQuery =
        from p in products
        select p;

    Console.WriteLine("The list of products:");
    foreach (Product product in productsQuery)
    {
        Console.WriteLine(product.Name);
    }

    IQueryable<Product> redProducts = productsQuery
        .Where(p => p.Color == "Red")
        .Select(p => p);

    Console.WriteLine("");
    Console.WriteLine("The list of red products:");
    foreach (Product redProduct in redProducts)
    {
        Console.WriteLine(redProduct.Name);
    }

}
```

PROCEDIMIENTO

1. EJEMPLOS DE SINTAXIS DE EXPRESIONES DE CONSULTA: PROYECCIÓN

Los ejemplos de este tema muestran cómo usar el **método Select** y las palabras **clave From**, para consultar el modelo AdventureWorks Sales con la sintaxis de las expresiones de consulta. From … es el equivalente basado en las consultas del **método SelectMany**. El modelo AdventureWorks Sales que se usa en estos ejemplos se crea a partir de las tablas Contact, Address, Product, SalesOrderHeader y SalesOrderDetail de la base de datos de ejemplo AdventureWorks.

Los ejemplos de este tema usan las instrucciones using/Imports siguientes:

```
using System;
using System.Data;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data.Objects;
using System.Globalization;
using System.Data.EntityClient;
using System.Data.SqlClient;
using System.Data.Common;
```

SELECT

Ejemplo

En el ejemplo siguiente se usa el método Select para devolver todas las filas de la tabla Product y mostrar los nombres de producto.

```
using (AdventureWorksEntities context = new AdventureWorksEntities())
{
    IQueryable<Product> productsQuery = from product in context.Products
                                        select product;

    Console.WriteLine("Product Names:");
    foreach (var prod in productsQuery)
    {
        Console.WriteLine(prod.Name);
    }
}
```

Ejemplo

El ejemplo siguiente utiliza Select para devolver una secuencia de nombres de producto solamente.

```
using (AdventureWorksEntities context = new AdventureWorksEntities())
{
    IQueryable<string> productNames =
        from p in context.Products
        select p.Name;

    Console.WriteLine("Product Names:");
    foreach (String productName in productNames)
```

```
    {
        Console.WriteLine(productName);
    }
}
```

Ejemplo

En el ejemplo siguiente se usa el método Select para proyectar las propiedades Product.Name y Product.ProductID en una secuencia de tipos anónimos.

```
using (AdventureWorksEntities context = new AdventureWorksEntities())
{
    var query =
        from product in context.Products
        select new
        {
            ProductId = product.ProductID,
            ProductName = product.Name
        };

    Console.WriteLine("Product Info:");
    foreach (var productInfo in query)
    {
        Console.WriteLine("Product Id: {0} Product name: {1} ",
            productInfo.ProductId, productInfo.ProductName);
    }
}
```

From … From … (SelectMany)

Ejemplo

En el ejemplo siguiente se usa From … From … (el equivalente del método SelectMany) para seleccionar todos los pedidos en los que TotalDue es inferior a 500,00.

```
decimal totalDue = 500.00M;
using (AdventureWorksEntities context = new AdventureWorksEntities())
{
    ObjectSet<Contact> contacts = context.Contacts;
    ObjectSet<SalesOrderHeader> orders = context.SalesOrderHeaders;

    var query =
        from contact in contacts
        from order in orders
        where contact.ContactID == order.Contact.ContactID
            && order.TotalDue < totalDue
        select new
        {
            ContactID = contact.ContactID,
            LastName = contact.LastName,
            FirstName = contact.FirstName,
            OrderID = order.SalesOrderID,
            Total = order.TotalDue
```

```
        };

    foreach (var smallOrder in query)
    {
        Console.WriteLine("Contact ID: {0} Name: {1}, {2} Order ID: {3} Total
Due: ${4} ",
            smallOrder.ContactID, smallOrder.LastName, smallOrder.FirstName,
            smallOrder.OrderID, smallOrder.Total);
    }
}
```

Ejemplo

En el ejemplo siguiente se utiliza From … From … (el equivalente del método SelectMany) para seleccionar todos los pedidos efectuados a partir del 1 de octubre de 2002.

```
using (AdventureWorksEntities context = new AdventureWorksEntities())
{
    ObjectSet<Contact> contacts = context.Contacts;
    ObjectSet<SalesOrderHeader> orders = context.SalesOrderHeaders;

    var query =
        from contact in contacts
        from order in orders
        where contact.ContactID == order.Contact.ContactID
            && order.OrderDate >= new DateTime(2002, 10, 1)
        select new
        {
            ContactID = contact.ContactID,
            LastName = contact.LastName,
            FirstName = contact.FirstName,
            OrderID = order.SalesOrderID,
            OrderDate = order.OrderDate
        };

    foreach (var order in query)
    {
        Console.WriteLine("Contact ID: {0} Name: {1}, {2} Order ID: {3} Order
date: {4:d} ",
            order.ContactID, order.LastName, order.FirstName,
            order.OrderID, order.OrderDate);
    }
}
```

Ejemplo

En el ejemplo siguiente se utiliza From … From … (el equivalente al método SelectMany) para seleccionar todos los pedidos en los que el total del pedido es superior a 10000,00 y se utiliza la asignación From para evitar que se solicite dos veces el total.

```
decimal totalDue = 10000.0M;
using (AdventureWorksEntities context = new AdventureWorksEntities())
{
```

```
    ObjectSet<Contact> contacts = context.Contacts;
    ObjectSet<SalesOrderHeader> orders = context.SalesOrderHeaders;

    var query =
        from contact in contacts
        from order in orders
        let total = order.TotalDue
        where contact.ContactID == order.Contact.ContactID
            && total >= totalDue
        select new
        {
            ContactID = contact.ContactID,
            LastName = contact.LastName,
            OrderID = order.SalesOrderID,
            total
        };

    foreach (var order in query)
    {
        Console.WriteLine("Contact ID: {0} Last name: {1} Order ID: {2} Total:
{3}",
            order.ContactID, order.LastName, order.OrderID, order.total);
    }
}
```

2. EJEMPLOS DE SINTAXIS DE EXPRESIÓN DE CONSULTAS: FILTRADO

Los ejemplos de este tema muestran cómo se usan los métodos Where y Where…Contains para consultar el modelo AdventureWorks Sales con la sintaxis de las expresiones de consulta. Tenga en cuenta que Where…Contains no se puede utilizar como parte de una consulta compilada.

El modelo AdventureWorks Sales que se usa en estos ejemplos se crea a partir de las tablas Contact, Address, Product, SalesOrderHeader y SalesOrderDetail de la base de datos de ejemplo AdventureWorks.

En los ejemplos de este tema se usan las instrucciones using/Imports siguientes:

```
using System;
using System.Data;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data.Objects;
using System.Globalization;
using System.Data.EntityClient;
using System.Data.SqlClient;
using System.Data.Common;
```

Where

Ejemplo

En el ejemplo siguiente se devuelven todos los pedidos en línea.

```
using (AdventureWorksEntities context = new AdventureWorksEntities())
{
    var onlineOrders =
        from order in context.SalesOrderHeaders
        where order.OnlineOrderFlag == true
        select new
        {
            SalesOrderID = order.SalesOrderID,
            OrderDate = order.OrderDate,
            SalesOrderNumber = order.SalesOrderNumber
        };

    foreach (var onlineOrder in onlineOrders)
    {
        Console.WriteLine("Order ID: {0} Order date: {1:d} Order number: {2}",
            onlineOrder.SalesOrderID,
            onlineOrder.OrderDate,
            onlineOrder.SalesOrderNumber);
    }
}
```

Ejemplo

En el ejemplo siguiente se devuelven los pedidos en los que la cantidad de pedido es superior a 2 e inferior a 6.

```
int orderQtyMin = 2;
int orderQtyMax = 6;
using (AdventureWorksEntities context = new AdventureWorksEntities())
{
    var query =
        from order in context.SalesOrderDetails
        where order.OrderQty > orderQtyMin && order.OrderQty < orderQtyMax
        select new
        {
            SalesOrderID = order.SalesOrderID,
            OrderQty = order.OrderQty
        };

    foreach (var order in query)
    {
        Console.WriteLine("Order ID: {0} Order quantity: {1}",
            order.SalesOrderID, order.OrderQty);
    }
}
```

Ejemplo

En el ejemplo siguiente se devuelven todos los productos de color rojo.

```
String color = "Red";
using (AdventureWorksEntities context = new AdventureWorksEntities())
{
    var query =
        from product in context.Products
        where product.Color == color
        select new
        {
            Name = product.Name,
            ProductNumber = product.ProductNumber,
            ListPrice = product.ListPrice
        };

    foreach (var product in query)
    {
        Console.WriteLine("Name: {0}", product.Name);
        Console.WriteLine("Product number: {0}", product.ProductNumber);
        Console.WriteLine("List price: ${0}", product.ListPrice);
        Console.WriteLine("");
    }
}
```

Ejemplo

En el ejemplo siguiente se utiliza el método Where para buscar los pedidos que se realizaron después del 1 de diciembre de 2003 y, a continuación, se utiliza la propiedad de navegación order.SalesOrderDetail para obtener los detalles de cada pedido.

```
using (AdventureWorksEntities context = new AdventureWorksEntities())
{
    IQueryable<SalesOrderHeader> query =
        from order in context.SalesOrderHeaders
        where order.OrderDate >= new DateTime(2003, 12, 1)
        select order;


    Console.WriteLine("Orders that were made after December 1, 2003:");
    foreach (SalesOrderHeader order in query)
    {
        Console.WriteLine("OrderID {0} Order date: {1:d} ",
            order.SalesOrderID, order.OrderDate);
        foreach (SalesOrderDetail orderDetail in order.SalesOrderDetails)
        {
            Console.WriteLine("  Product ID: {0} Unit Price {1}",
                orderDetail.ProductID, orderDetail.UnitPrice);
        }
    }
}
```

Where…Contains

Ejemplo

En el ejemplo siguiente se usa una matriz como parte de una cláusula Where…Contains para encontrar todos los productos que tienen un ProductModelID que coincide con un valor de la matriz.

```
using (AdventureWorksEntities AWEntities = new AdventureWorksEntities())
{
    int?[] productModelIds = {19, 26, 118};
    var products = from p in AWEntities.Products
                   where productModelIds.Contains(p.ProductModelID)
                   select p;
    foreach (var product in products)
    {
        Console.WriteLine("{0}: {1}", product.ProductModelID,
product.ProductID);
    }
}
```

Ejemplo

El ejemplo siguiente declara e inicializa las matrices en una cláusula Where…Contains para encontrar todos los productos que tienen un ProductModelID o Size que coinciden con los valores de las matrices.

```
using (AdventureWorksEntities AWEntities = new AdventureWorksEntities())
{
    var products = from p in AWEntities.Products
                   where (new int?[] { 19, 26, 18 }).Contains(p.ProductModelID)
||
                         (new string[] { "L", "XL" }).Contains(p.Size)
                   select p;
    foreach (var product in products)
    {
        Console.WriteLine("{0}: {1}, {2}", product.ProductID,
                                           product.ProductModelID,
                                           product.Size);
    }
}
```

3. EJEMPLOS DE SINTAXIS DE EXPRESIONES DE CONSULTA: ORDENACIÓN

Los ejemplos de este tema muestran cómo se usan los métodos OrderBy y OrderByDescending para consultar el modelo AdventureWorks Sales con la sintaxis de las expresiones de consulta. El modelo AdventureWorks Sales que se usa en estos ejemplos se crea a partir de las tablas Contact, Address, Product, SalesOrderHeader y SalesOrderDetail de la base de datos de ejemplo AdventureWorks.

En los ejemplos de este tema se usan las instrucciones using/Imports siguientes:

```
using System;
```

```
using System.Data;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data.Objects;
using System.Globalization;
using System.Data.EntityClient;
using System.Data.SqlClient;
using System.Data.Common;
```

OrderBy

Ejemplo

En el ejemplo siguiente se utiliza OrderBy para devolver una lista de contactos organizados por apellido.

```
using (AdventureWorksEntities context = new AdventureWorksEntities())
{
    IQueryable<Contact> sortedNames =
        from n in context.Contacts
        orderby n.LastName
        select n;

    Console.WriteLine("The sorted list of last names:");
    foreach (Contact n in sortedNames)
    {
        Console.WriteLine(n.LastName);
    }
}
```

Ejemplo

En el ejemplo siguiente se utiliza OrderBy para ordenar una lista de contactos por longitud del apellido.

```
using (AdventureWorksEntities context = new AdventureWorksEntities())
{
    IQueryable<Contact> sortedNames =
        from n in context.Contacts
        orderby n.LastName.Length
        select n;

    Console.WriteLine("The sorted list of last names (by length):");
    foreach (Contact n in sortedNames)
    {
        Console.WriteLine(n.LastName);
    }
}
```

OrderByDescending

Ejemplo

En el ejemplo siguiente se utiliza orderby… descending (Order By … Descending) en Visual Basic, que es equivalente al método OrderByDescending, para ordenar el precio de venta de mayor a menor.

```
using (AdventureWorksEntities context = new AdventureWorksEntities())
{
    IQueryable<Decimal> sortedPrices =
        from p in context.Products
        orderby p.ListPrice descending
        select p.ListPrice;

    Console.WriteLine("The list price from highest to lowest:");
    foreach (Decimal price in sortedPrices)
    {
        Console.WriteLine(price);
    }
}
```

ThenBy

Ejemplo

En el ejemplo siguiente se utiliza OrderBy y ThenBy para devolver una lista de contactos ordenados por apellido y luego por nombre.

```
using (AdventureWorksEntities context = new AdventureWorksEntities())
{
    IQueryable<Contact> sortedContacts =
        from contact in context.Contacts
        orderby contact.LastName, contact.FirstName
        select contact;

    Console.WriteLine("The list of contacts sorted by last name then by first
name:");
    foreach (Contact sortedContact in sortedContacts)
    {
        Console.WriteLine(sortedContact.LastName + ", " +
sortedContact.FirstName);
    }
}
```

ThenByDescending

Ejemplo

En el ejemplo siguiente se utiliza OrderBy… Descending, que es equivalente al método ThenByDescending, para ordenar una lista de productos, primero por nombre y después por precio de venta, de mayor a menor.

```
using (AdventureWorksEntities context = new AdventureWorksEntities())
```

```
{
    IQueryable<Product> query =
        from product in context.Products
        orderby product.Name, product.ListPrice descending
        select product;

    foreach (Product product in query)
    {
        Console.WriteLine("Product ID: {0} Product Name: {1} List Price {2}",
            product.ProductID,
            product.Name,
            product.ListPrice);
    }
}
```

4. EJEMPLOS DE SINTAXIS DE EXPRESIONES DE CONSULTA: OPERADORES DE AGREGADO

Los ejemplos de este tema muestran cómo se utilizan los métodos Average, Count, Max, Min y Sum para consultar el modelo AdventureWorks Sales con la sintaxis de las expresiones de consulta. El modelo AdventureWorks Sales que se usa en estos ejemplos se compila a partir de las tablas Contact, Address, Product, SalesOrderHeader y SalesOrderDetail en la base de datos de ejemplo AdventureWorks.

En los ejemplos de este tema se usan las instrucciones using/Imports siguientes:

```
using System;
using System.Data;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data.Objects;
using System.Globalization;
using System.Data.EntityClient;
using System.Data.SqlClient;
using System.Data.Common;
```

Average

Ejemplo

En el ejemplo siguiente se utiliza el método Average para encontrar el precio de venta promedio de los productos de cada estilo.

```
using (AdventureWorksEntities context = new AdventureWorksEntities())
{
    ObjectSet<Product> products = context.Products;

    var query = from product in products
                group product by product.Style into g
                select new
                {
```

```
                    Style = g.Key,
                    AverageListPrice =
                        g.Average(product => product.ListPrice)
                };

    foreach (var product in query)
    {
        Console.WriteLine("Product style: {0} Average list price: {1}",
            product.Style, product.AverageListPrice);
    }
}
```

Ejemplo

En el ejemplo siguiente se utiliza Average para obtener el importe total a pagar promedio para cada identificador de contacto.

```
using (AdventureWorksEntities context = new AdventureWorksEntities())
{
    ObjectSet<SalesOrderHeader> orders = context.SalesOrderHeaders;

    var query =
        from order in orders
        group order by order.Contact.ContactID into g
        select new
        {
            Category = g.Key,
            averageTotalDue = g.Average(order => order.TotalDue)
        };

    foreach (var order in query)
    {
        Console.WriteLine("ContactID = {0} \t Average TotalDue = {1}",
            order.Category, order.averageTotalDue);
    }
}
```

Ejemplo

En el ejemplo siguiente se utiliza el método Average para obtener el pedido con el importe total promedio a pagar de cada contacto.

```
using (AdventureWorksEntities context = new AdventureWorksEntities())
{
    ObjectSet<SalesOrderHeader> orders = context.SalesOrderHeaders;

    var query =
        from order in orders
        group order by order.Contact.ContactID into g
        let averageTotalDue = g.Average(order => order.TotalDue)
        select new
        {
```

```
            Category = g.Key,
            CheapestProducts =
                g.Where(order => order.TotalDue == averageTotalDue)
        };

    foreach (var orderGroup in query)
    {
        Console.WriteLine("ContactID: {0}", orderGroup.Category);
        foreach (var order in orderGroup.CheapestProducts)
        {
            Console.WriteLine("Average total due for SalesOrderID {1} is: {0}",
                order.TotalDue, order.SalesOrderID);
        }
        Console.Write("\n");
    }
}
```

Count

Ejemplo

En el ejemplo siguiente se utiliza el método Count para devolver una lista de identificadores de contactos y el número de pedidos que tiene cada uno.

```
using (AdventureWorksEntities context = new AdventureWorksEntities())
{
    ObjectSet<Contact> contacts = context.Contacts;

    //Can't find field SalesOrderContact
    var query =
        from contact in contacts
        select new
        {
            CustomerID = contact.ContactID,
            OrderCount = contact.SalesOrderHeaders.Count()
        };

    foreach (var contact in query)
    {
        Console.WriteLine("CustomerID = {0} \t OrderCount = {1}",
            contact.CustomerID,
            contact.OrderCount);
    }
}
```

Ejemplo

En el ejemplo siguiente se agrupan los productos por colores y se utiliza Count para devolver el número de productos de cada grupo de color.

```
using (AdventureWorksEntities context = new AdventureWorksEntities())
{
    ObjectSet<Product> products = context.Products;
```

```
    var query =
        from product in products
        group product by product.Color into g
        select new { Color = g.Key, ProductCount = g.Count() };

    foreach (var product in query)
    {
        Console.WriteLine("Color = {0} \t ProductCount = {1}",
            product.Color,
            product.ProductCount);
    }
}
```

Max

Ejemplo

En el ejemplo siguiente se utiliza el método Max para obtener el mayor importe total a pagar de cada identificador de contacto.

```
using (AdventureWorksEntities context = new AdventureWorksEntities())
{
    ObjectSet<SalesOrderHeader> orders = context.SalesOrderHeaders;

    var query =
        from order in orders
        group order by order.Contact.ContactID into g
        select new
        {
            Category = g.Key,
            maxTotalDue =
                g.Max(order => order.TotalDue)
        };

    foreach (var order in query)
    {
        Console.WriteLine("ContactID = {0} \t Maximum TotalDue = {1}",
            order.Category, order.maxTotalDue);
    }
}
```

Ejemplo

En el ejemplo siguiente se utiliza el método Max para obtener el pedido con el mayor importe total a pagar de cada identificador de contacto.

```
using (AdventureWorksEntities context = new AdventureWorksEntities())
{
    ObjectSet<SalesOrderHeader> orders = context.SalesOrderHeaders;

    var query =
        from order in orders
        group order by order.Contact.ContactID into g
```

```
        let maxTotalDue = g.Max(order => order.TotalDue)
        select new
        {
            Category = g.Key,
            CheapestProducts =
                g.Where(order => order.TotalDue == maxTotalDue)
        };

    foreach (var orderGroup in query)
    {
        Console.WriteLine("ContactID: {0}", orderGroup.Category);
        foreach (var order in orderGroup.CheapestProducts)
        {
            Console.WriteLine("MaxTotalDue {0} for SalesOrderID {1}: ",
                order.TotalDue,
                order.SalesOrderID);
        }
        Console.Write("\n");
    }
}
```

Min

Ejemplo

En el ejemplo siguiente se utiliza el método Min para obtener el menor importe total a pagar de cada identificador de contacto.

```
using (AdventureWorksEntities context = new AdventureWorksEntities())
{
    ObjectSet<SalesOrderHeader> orders = context.SalesOrderHeaders;

    var query =
        from order in orders
        group order by order.Contact.ContactID into g
        select new
        {
            Category = g.Key,
            smallestTotalDue =
                g.Min(order => order.TotalDue)
        };

    foreach (var order in query)
    {
        Console.WriteLine("ContactID = {0} \t Minimum TotalDue = {1}",
            order.Category, order.smallestTotalDue);
    }
}
```

Ejemplo

En el ejemplo siguiente se utiliza el método Min para obtener el pedido con el menor importe total a pagar de cada contacto.

```
using (AdventureWorksEntities context = new AdventureWorksEntities())
{
    ObjectSet<SalesOrderHeader> orders = context.SalesOrderHeaders;

    var query =
        from order in orders
        group order by order.Contact.ContactID into g
        let minTotalDue = g.Min(order => order.TotalDue)
        select new
        {
            Category = g.Key,
            smallestTotalDue =
                g.Where(order => order.TotalDue == minTotalDue)
        };


    foreach (var orderGroup in query)
    {
        Console.WriteLine("ContactID: {0}", orderGroup.Category);
        foreach (var order in orderGroup.smallestTotalDue)
        {
            Console.WriteLine("Mininum TotalDue {0} for SalesOrderID {1}: ",
                order.TotalDue,
                order.SalesOrderID);
        }
        Console.Write("\n");
    }
}
```

Sum

Ejemplo

En el ejemplo siguiente se utiliza el método Sum para obtener el menor importe total a pagar de cada identificador de contacto.

```
using (AdventureWorksEntities context = new AdventureWorksEntities())
{
    ObjectSet<SalesOrderHeader> orders = context.SalesOrderHeaders;

    var query =
        from order in orders
        group order by order.Contact.ContactID into g
        select new
        {
            Category = g.Key,
            TotalDue = g.Sum(order => order.TotalDue)
        };
```

```
    foreach (var order in query)
    {
        Console.WriteLine("ContactID = {0} \t TotalDue sum = {1}",
            order.Category, order.TotalDue);
    }
}
```

5. EJEMPLOS DE SINTAXIS DE CONSULTA BASADA EN MÉTODOS: OPERADORES DE ELEMENTOS

Los ejemplos de este tema muestran cómo se usa el método First para consultar el modelo AdventureWorks Sales con la sintaxis de las consultas basadas en métodos. El modelo AdventureWorks Sales que se usa en estos ejemplos se crea a partir de las tablas Contact, Address, Product, SalesOrderHeader y SalesOrderDetail en la base de datos de ejemplo de AdventureWorks.

El ejemplo de este tema usa las siguientes instrucciones using/Imports:

```
using System;
using System.Data;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data.Objects;
using System.Globalization;
using System.Data.EntityClient;
using System.Data.SqlClient;
using System.Data.Common;
```

First

Ejemplo

En el ejemplo siguiente se usa el método First para buscar la primera dirección de correo electrónico que empieza por "caroline".

```
string name = "caroline";
using (AdventureWorksEntities context = new AdventureWorksEntities())
{
    ObjectSet<Contact> contacts = context.Contacts;

    Contact query = contacts.First(contact =>
        contact.EmailAddress.StartsWith(name));

    Console.WriteLine("An email address starting with 'caroline': {0}",
        query.EmailAddress);
}
```

6. EJEMPLOS DE SINTAXIS DE CONSULTA BASADA EN MÉTODOS: CONVERSIÓN

Los ejemplos de este tema muestran cómo se usan los métodos ToArray<TSource>, ToDictionary y ToList<TSource> para consultar el modelo AdventureWorks Sales usando la sintaxis de las consultas basadas en métodos. El modelo AdventureWorks Sales que se usa en estos ejemplos se crea a partir de las tablas Contact, Address, Product, SalesOrderHeader y SalesOrderDetail de la base de datos de ejemplo AdventureWorks.

Los ejemplos de este tema usan las instrucciones using/Imports siguientes:

```
using System;
using System.Data;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data.Objects;
using System.Globalization;
using System.Data.EntityClient;
using System.Data.SqlClient;
using System.Data.Common;
```

ToArray

Ejemplo

En el ejemplo siguiente se usa el método ToArray<TSource> para evaluar de forma inmediata una secuencia en una matriz.

```
using (AdventureWorksEntities context = new AdventureWorksEntities())
{
    ObjectSet<Product> products = context.Products;

    Product[] prodArray = (
        from product in products
        orderby product.ListPrice descending
        select product).ToArray();

    Console.WriteLine("Every price from highest to lowest:");
    foreach (Product product in prodArray)
    {
        Console.WriteLine(product.ListPrice);
    }
}
```

ToDictionary

Ejemplo

En el ejemplo siguiente se utiliza el método ToDictionary para evaluar de forma inmediata una secuencia y una expresión de clave relacionada en un diccionario.

```
using (AdventureWorksEntities context = new AdventureWorksEntities())
{
    ObjectSet<Product> products = context.Products;

    Dictionary<String, Product> scoreRecordsDict = products.
            ToDictionary(record => record.Name);

    Console.WriteLine("Top Tube's ProductID: {0}",
            scoreRecordsDict["Top Tube"].ProductID);
}
```

ToList

Ejemplo

En el ejemplo siguiente se utiliza el método ToList<TSource> para evaluar inmediatamente una secuencia en List<T>, donde T es de tipo DataRow.

```
using (AdventureWorksEntities context = new AdventureWorksEntities())
{
    ObjectSet<Product> products = context.Products;

    List<Product> query =
        (from product in products
         orderby product.Name
         select product).ToList();

    Console.WriteLine("The product list, ordered by product name:");
    foreach (Product product in query)
    {
        Console.WriteLine(product.Name.ToLower(CultureInfo.InvariantCulture));
    }
}
```