# KếtLINQ to SQL:

LINQ to SQL là một thành phần của. NET Framework (3.5 trở lên) cung cấp cơ sở hạ tầng thời gian thực để giúp quản lý các dữ liệu quan hệ như các đối tượng.

*Chú ý*: Các dữ liệu quan hệ sẽ hiện diện như là một danh sách của các bảng 2 chiều (các quan hệ hoặc các flat file), nơi mà các cột bảng liên quan đến nhau. Để sử dụng LINQ to SQL một cách có hiệu quả, bạn cần phải làm quen với một số khái niệm cơ bản về cơ sở dữ liệu quan hệ.

Trong LINQ to SQL, mô hình dữ liệu của cơ sở dữ liệu quan hệ được ánh xạ tới mô hình đối tượng được mô tả trong ngôn ngữ lập trình của lập trình viên. Khi các ứng dụng khởi chạy, LINQ to SQL dịch các truy vấn LINQ trong các mô hình đối tượng thành các truy vấn SQL và gửi chúng đến CSDL để thực thi. Khi cơ sở dữ liệu trả về kết quả, LINQ to SQL chuyển chúng trở lại thành các đối tượng mà bạn có thể sử dụng cho ngôn ngữ lập trình của mình.

Các lập trình viên sử dụng Visual Studio thông thường sẽ sử dụng các Object Relational Designer (thết kế đối tượng quan hệ), thứ có thể cung cấp giao diện người dùng giúp họ thực thi nhiều tính năng của LINQ to SQL.

LINQ to SQL là một phần của dòng công nghệ ADO.NET. Nó được xây dựng dựa trên các dịch vụ được cung cấp bởi mô hình ADO.NET provider. Do vậy, bạn có thể pha trộn code LINQ to SQL với các ứng dụng ADO.NET sẵn có và sử dụng các giải pháp ADO.NET cho LINQ to SQL. Ví dụ minh họa sau cung cấp cái nhìn cấp cao về các mối quan hệ



## Kết nối:

Bạn có thể cung cấp (cho hệ thống) một kết nối ADO.NET đã tồn tại khi bạn tạo ra một LINQ to SQL DataContext. Tất cả các hoạt động dựa vào DataContext (bao gồm cả các truy vấn) đều sử dụng kết nối đã được cung cấp này. Nếu kết nối đã được mở, LINQ to SQL vẫn sẽ để nó như vậy khi bạn đã xong việc với nó.

string connString = @"Data Source=.\SQLEXPRESS;AttachDbFilename=c:\northwind.mdf;

Integrated Security=True; Connect Timeout=30; User Instance=True";

SqlConnection nwindConn = new SqlConnection(connString);

nwindConn.Open();

Northwnd interop\_db = new Northwnd(nwindConn);

SqlTransaction nwindTxn = nwindConn.BeginTransaction();

try

{

SqlCommand cmd = new SqlCommand(

"UPDATE Products SET QuantityPerUnit = 'single item' WHERE ProductID = 3");

cmd.Connection = nwindConn;

cmd.Transaction = nwindTxn;

cmd.ExecuteNonQuery();

interop\_db.Transaction = nwindTxn;

Product prod1 = interop\_db.Products

.First(p => p.ProductID == 4);

Product prod2 = interop\_db.Products

.First(p => p.ProductID == 5);

prod1.UnitsInStock -= 3;

prod2.UnitsInStock -= 5;

interop\_db.SubmitChanges();

nwindTxn.Commit();

}

catch (Exception e)

{

Console.WriteLine(e.Message);

Console.WriteLine("Error submitting changes... all changes rolled back.");

}

nwindConn.Close();

Bạn luôn luôn có thể truy cập vào kết nối và đóng nó bằng cách sử dụng các đặc tính của Connection như sau:

db.Connection.Close();

## Giao dịch:

Bạn có thể cung cấp cho DataContext bằng giao dịch từ CSDL của chính mình khi các ứng dụng của bạn đã khởi động khả năng giao dịch và bạn muốn DataContext tham gia vào.

Phương pháp được thông dụng nhất để thực hiện giao dịch .NET Framework là sử dụng những đối tượng TransactionScope. Bằng cách sử dụng phương pháp tiếp cận này, bạn có thể thực hiện các giao dịch trên nhiều cơ sở dữ liệu và các tài nguyên bộ nhớ khác. TransactionScope yêu cầu rất ít tài nguyên để khởi động. Chúng chỉ bắt đầu phân bổ các giao dịch chỉ khi có nhiều kết nối trong phạm vi giao dịch.

using (TransactionScope ts = new TransactionScope())

{

db.SubmitChanges();

ts.Complete();

}

Bạn không thể sử dụng phương pháp tiếp cận này cho tất cả các cơ sở dữ liệu. Ví dụ, các kết nối SqlClient không thể hoạt động với các giao dịch hệ thống vì nó hoạt động dựa trên một máy chủ SQL Server 2000. Thay vào đó, nó sẽ tự động tham gia vào một giao dịch đầy đủ và đã được phân phối chức năng liên quan ngay khi thấy giao dịch này được sử dụng.

## Lệnh gọi SQL trực tiếp:

Đôi khi bạn có thể gặp tình huống mà DataContext không đủ khả năng để truy vấn hoặc gửi đi các thay đổi theo đúng yêu cầu chuyên môn mà bạn muốn thực hiện. Trong những trường hợp đó, bạn có thể sử dụng các phương pháp ExecuteQuery để thực hiện trực tiếp các lệnh SQL cho cơ sở dữ liệu và chuyển đổi kết quả truy vấn vào các đối tượng.

Ví dụ, giả định rằng dữ liệu của các Customer nằm trên hai bảng (Customer 1 và Customer 2). truy vấn sau đây sẽ trả về một kết quả cho đối tượng Customer:

IEnumerable<Customer> results = db.ExecuteQuery<Customer>(

@"select c1.custid as CustomerID, c2.custName as ContactName

from customer1 as c1, customer2 as c2

where c1.custid = c2.custid"

);

Chỉ cần các tên cột trong các kết quả phù hợp với các thuộc tính cột trong lớp thực thể của bạn, LINQ to SQL sẽ tạo ra được các đối tượng từ truy vấn SQL.

## Các tham số:

Phương pháp ExecuteQuery chấp nhận các tham số. Đoạn code sau đây thực thi truy vấn có tham số:

IEnumerable<Customer> results = db.ExecuteQuery<Customer>(

"select contactname from customers where city = {0}",

"London"

);

LINQ to SQL sẽ tạo ra được các đối tượng từ truy vấn SQL.

## Cách kết nối một CSDL (LINQ to SQL):

DataContext là đường dẫn chính để bạn kết nối, lấy dữ liệu cho các đối tượng, và cập nhật dữ liệu mới trên một CSDL. Bạn chỉ cần sử dụng DataContext tương tự như khi bạn sử dụng ADO.NET SqlConnection. Trong thực tế, các DataContext được khởi tạo với một kết nối hoặc chuỗi kết nối mà bạn cung cấp.

Mục đích của DataContext là để dịch các yêu cầu của bạn cho các đối tượng thành các truy vấn của SQL để làm việc với các cơ sở dữ liệu, và sau đó thu thập các đối tượng từ các kết quả. DataContext kết hợp với LINQ bằng cách lấy cùng những toán tử như nhau để làm chuẩn, chẳng hạn như “Where” và “Select”.

Trong ví dụ sau, DataContext được sử dụng để kết nối với cơ sở dữ liệu mẫu Northwind và để lấy thông tin của những khách hàng là người sống ở London

// DataContext takes a connection string.

DataContext db = new DataContext(@"c:\Northwind.mdf");

// Get a typed table to run queries.

Table<Customer> Customers = db.GetTable<Customer>();

// Query for customers from London.

var query =

from cust in Customers

where cust.City == "London"

select cust;

foreach (var cust in query)

Console.WriteLine("id = {0}, City = {1}", cust.CustomerID, cust.City);

Mỗi bảng trong CSDL được biểu điễn bởi một Table trong danh sách cho phép truy vấn bằng phương thức GetTable bằng cách sử dụng lớp thực thể để xác định nó.

Phương pháp tốt nhất là khai báo một kiểu DataContext thay vì dựa trên lớp DataContext cơ bản và phương pháp GetTable. Kiểu DataContext khai báo tập hợp tất cả các Table, như một thành phần của ngữ cảnh theo ví dụ sau:

public partial class Northwind : DataContext

{

public Table<Customer> Customers;

public Table<Order> Orders;

public Northwind(string connection) : base(connection) { }

}

Sau đó bạn có thể thực hiên truy vấn khách hàng từ London đơn giản hơn bằng cách sau:

Northwnd db = new Northwnd(@"c:\Northwnd.mdf");

var query =

from cust in db.Customers

where cust.City == "London"

select cust;

foreach (var cust in query)

Console.WriteLine("id = {0}, City = {1}", cust.CustomerID,

cust.City);

LINQ to SQL sẽ tạo ra được các đối tượng từ truy vấn SQL.

## Cách tự động tạo ra một CSDL (LINQ to SQL):

Trong LINQ to SQL, các đối tượng được ánh xạ vào một CSDL quan hệ. Sự ánh xạ được kích hoạt bằng cách ánh xạ các đặc tính tượng trưng hoặc sử dụng các tệp ánh xạ ngoài để mô tả cấu trúc của CSDL quan hệ. Cả 2 phương pháp này đều sẽ cung cấp đầy đủ thông tin về CSDL quan hệ để bạn có thể tạo ra một phiên bản của CSDL sử dụng phương thức DataContext.CreateDatabase.

Phương thức DataContext.CreateDatabase tạo một bản sao của cơ sở dữ liệu nằm trong phạm vi thông tin được mã hóa trong mô hình đối tượng. Ánh xạ tệp và thuộc tính từ mô hình đối tượng của bạn có thể sẽ không mã hóa mọi thứ về cấu trúc của cơ sở dữ liệu hiện có. Thông tin ánh xạ sẽ không thể bao gồm nội dung của các hàm do người dùng xác định, các hàm, thủ tục lưu trữ, trigger hoặc khả năng kiểm tra các ràng buộc.

Bạn có thể sử dụng phương thức DataContext. CreateDatabase trong mọi trường hợp, đặc biệt là nếu bên cung cấp dữ liệu có Microsoft SQL Server 2008 khả dụng. Các kịch bản điển hình bao gồm:

* Bạn đang xây dựng một ứng dụng có thể tự động cài đặt trên hệ thống của khách hàng.
* Bạn đang xây dựng một ứng dụng khách cần cơ sở dữ liệu cục bộ để lưu các trạng thái ngoại tuyến của nó.

Bạn cũng có thể sử dụng phương thức DataContext.CreateDatabase với SQL Server bằng cách sử dụng tệp .mdf hoặc tên thay thế, tùy theo chuỗi kết nối của bạn. LINQ to SQL sử dụng chuỗi kết nối để xác định CSDL nguồn và CSDL nào được tạo.

Ví dụ: Đoạn code sau đây cung cấp một ví dụ về cách bạn sẽ tạo ra một cơ sở dữ liệu mới có tên MyDVDs.mdf.

public class MyDVDs : DataContext

{

public Table<DVD> DVDs;

public MyDVDs(string connection) : base(connection) { }

}

[Table(Name = "DVDTable")]

public class DVD

{

[Column(IsPrimaryKey = true)]

public string Title;

[Column]

public string Rating;

}

Bạn cũng có thể sử dụng mô hình đối tượng để tạo CSDL như sau:

public void CreateDatabase()

{

MyDVDs db = new MyDVDs("c:\\mydvds.mdf");

db.CreateDatabase();

}

Khi bạn xây dựng một ứng dụng có thể tự động cài đặt lên hệ thống của khách hàng, hãy xem liệu CSDL của ứng dụng đã tồn tại hay chưa và xóa nó trước khi tạo một CSDL mới. Lớp DataContext cung cấp các phương thức DatabaseExists và DeleteDatabase để giúp bạn trong quá trình này.

public void CreateDatabase2()

{

MyDVDs db = new MyDVDs(@"c:\mydvds.mdf");

if (db.DatabaseExists())

{

Console.WriteLine("Deleting old database...");

db.DeleteDatabase();

}

db.CreateDatabase();

}

## Bạn có thể làm gì với LINQ to SQL:

LINQ to SQL cung cấp tất cả các từ khóa cần thiết dành cho một lập trình viên SQL. Bạn có thể truy vấn dữ liệu và thêm, sửa, xóa các dữ liệu trong bảng.

### Lựa chọn (Selecting):

Là viết một truy vấn LINQ trong ngôn ngữ lập trình của mình và sau đó xử lý truy vấn để lấy các kết quả mong muốn. LINQ to SQL tự động dịch tất cả các hoạt động cần thiết thành các truy vấn SQL cần thiết mà bạn có thể hiểu được.

Ví dụ sau đây sẽ thực hiện lấy tên công ty của một khách hàng đến từ London và hiển thị nó lên màn hình Console

// Northwnd inherits from System.Data.Linq.DataContext.

Northwnd nw = new Northwnd(@"northwnd.mdf");

// or, if you are not using SQL Server Express

// Northwnd nw = new Northwnd("Database=Northwind;Server=server\_name;Integrated Security=SSPI");

var companyNameQuery =

from cust in nw.Customers

where cust.City == "London"

select cust.CompanyName;

foreach (var customer in companyNameQuery)

{

Console.WriteLine(customer);

}

### Thêm (Inserting):

Để thực hiện một truy vấn Insert như trong SQL, chỉ cần thêm các đối tượng vào mô hình đối tượng bạn đã tạo, và gọi lệnh SubmitChanges trên DataContext.

Trong ví dụ sau, một khách hàng mới và các thông tin về các khách hàng sẽ được thêm vào bảng Customers bằng cách sử dụng lệnh InsertOnSubmit.

// Northwnd inherits from System.Data.Linq.DataContext.

Northwnd nw = new Northwnd(@"northwnd.mdf");

Customer cust = new Customer();

cust.CompanyName = "SomeCompany";

cust.City = "London";

cust.CustomerID = "98128";

cust.PostalCode = "55555";

cust.Phone = "555-555-5555";

nw.Customers.InsertOnSubmit(cust);

// At this point, the new Customer object is added in the object model.

// In LINQ to SQL, the change is not sent to the database until

// SubmitChanges is called.

nw.SubmitChanges();

### Cập nhật (Updating):

Để Update cơ sở dữ liệu, đầu tiên bạn cần truy xuất mục cần thay đổi và chỉnh sửa trực tiếp trong mô hình đối tượng. Sau khi bạn đã sửa đổi đối tượng, hãy gọi SendChanges trên DataContext để cập nhật vào CSDL.

Trong ví dụ sau, tất cả các khách hàng đến từ London đều được truy xuất. Sau đó, tên của thành phố được đổi từ "London" sang "London - Metro". Cuối cùng, SendChanges được gọi để gửi các thay đổi đến CSDL.

Northwnd nw = new Northwnd(@"northwnd.mdf");

var cityNameQuery =

from cust in nw.Customers

where cust.City.Contains("London")

select cust;

foreach (var customer in cityNameQuery)

{

if (customer.City == "London")

{

customer.City = "London - Metro";

}

}

nw.SubmitChanges();

### Xóa (Deleting):

Để Delete một mục, hãy xóa mục đó khỏi danh sách đối tượng và sau đó gọi SendChanges trên DataContext để thực hiện thay đổi.

Lưu ý: LINQ to SQL không thể nhận ra các ràng buộc liên quan khi xóa một mục. Nếu bạn muốn xóa một hàng trong bảng mà có các ràng buộc liên quan đến nó hãy xem lại phần “Chỉ định lưu trữ Thực hiện thủ tục Update, Insert, và delete”.

Trong ví dụ sau, khách hàng có “CustomerID” là “98128” được lấy từ CSDL. Sau đó, sau khi xác nhận rằng hàng của khách hàng đã được truy xuất vào đối tượng quan hệ, DeleteOnSubmit được gọi để xóa đối tượng đó khỏi danh sách. Cuối cùng, SendChanges được gọi để tạo yêu cầu xóa tại CSDL.

Northwnd nw = new Northwnd(@"northwnd.mdf");

var deleteIndivCust =

from cust in nw.Customers

where cust.CustomerID == "98128"

select cust;

if (deleteIndivCust.Count() > 0)

{

nw.Customers.DeleteOnSubmit(deleteIndivCust.First());

nw.SubmitChanges();

}

## Thủ tục lưu trữ (Stored Procedures):

LINQ to SQL sử dụng các phương thức trong mô hình đối tượng để thể hiện các thủ tục lưu trữ vào CSDL. Bạn sẽ chỉ định các phương thức và các thủ tục lưu trữ bằng cách áp dụng thuộc tính FunctionAttribution và thuộc tính ParameterAttribution nếu cần thiết.

Các Lập trình viên sử dụng Visual Studio thường sử dụng Object Relational Designer để ánh xạ các thủ tục lưu trữ.

### Trả về RowSets:

Khi bạn thực hiện một Stored Procedure trả về một rowset, bạn phải sử dụng một lớp kết quả lưu trữ giá trị trả về từ Stored Procedure đó.

Ví dụ sau trả về một rowset từ CSDL và có một tham số đầu vào để lọc kết quả, tham số đầu vào sẽ lọc ra những khách hàng có “London” là thành phố. Ví dụ có chứa lớp Enumerable CustomersByCityResult

CREATE PROCEDURE [dbo].[Customers By City]

(@param1 NVARCHAR(20))

AS

BEGIN

-- SET NOCOUNT ON added to prevent extra result sets from

-- interfering with SELECT statements.

SET NOCOUNT ON;

SELECT CustomerID, ContactName, CompanyName, City from Customers

as c where c.City=@param1

END

[Function(Name="dbo.Customers By City")]

public ISingleResult<CustomersByCityResult> CustomersByCity([Parameter(DbType="NVarChar(20)")] string param1)

{

IExecuteResult result = this.ExecuteMethodCall(this, ((MethodInfo)(MethodInfo.GetCurrentMethod())), param1);

return ((ISingleResult<CustomersByCityResult>)(result.ReturnValue));

}

// Call the stored procedure.

void ReturnRowset()

{

Northwnd db = new Northwnd(@"c:\northwnd.mdf");

ISingleResult<CustomersByCityResult> result =

db.CustomersByCity("London");

foreach (CustomersByCityResult cust in result)

{

Console.WriteLine("CustID={0}; City={1}", cust.CustomerID,

cust.City);

}

}

### Sử dụng Stored Procedures với tham số đầu vào:

LINQ to SQL ánh xạ các tham số đầu ra thành các tham số tham chiếu và khai báo kiểu dữ liệu của các tham số này là Nullable.

Ví dụ sau nhận một tham số đầu vào (ID khách hàng) và trả về một tham số (tổng doanh số cho khách hàng đó).

CREATE PROCEDURE [dbo].[CustOrderTotal]

@CustomerID nchar(5),

@TotalSales money OUTPUT

AS

SELECT @TotalSales = SUM(OD.UNITPRICE\*(1-OD.DISCOUNT) \* OD.QUANTITY)

FROM ORDERS O, "ORDER DETAILS" OD

where O.CUSTOMERID = @CustomerID AND O.ORDERID = OD.ORDERID

[Function(Name="dbo.CustOrderTotal")]

[return: Parameter(DbType="Int")]

public int CustOrderTotal([Parameter(Name="CustomerID", DbType="NChar(5)")] string customerID, [Parameter(Name="TotalSales", DbType="Money")] ref System.Nullable<decimal> totalSales)

{

IExecuteResult result = this.ExecuteMethodCall(this, ((MethodInfo)(MethodInfo.GetCurrentMethod())), customerID, totalSales);

totalSales = ((System.Nullable<decimal>)(result.GetParameterValue(1)));

return ((int)(result.ReturnValue));

}

// Call the stored procedure.

Northwnd db = new Northwnd(@"c:\northwnd.mdf");

decimal? totalSales = 0;

db.CustOrderTotal("alfki", ref totalSales);

Console.WriteLine(totalSales);

### Sử dụng Stored Procedure ánh xạ cho nhiều dạng kết quả:

Khi một Stored Procedures có thể trả về nhiều dạng kết quả, kiểu trả về không thể phụ thuộc vào một phép chiếu. Mặc dù LINQ to SQL có thể tạo ra tất cả các loại phép chiếu có thể, nhưng nó không thể biết thứ tự chúng sẽ được trả về. Mặc dù LINQ to SQL có thể giải quyết tất cả các phép chiếu có thể, nhưng nó cũng không thể biết được thứ tự trả về của chúng

Trong ví dụ sau, dạng kết qủa phụ thuộc vào đầu vào (shape = 1 hay shape = 2). Bạn sẽ không biết rằng phép chiếu nào sẽ hoàn thành trước.

CREATE PROCEDURE VariableResultShapes(@shape int)

AS

if(@shape = 1)

select CustomerID, ContactTitle, CompanyName from customers

else if(@shape = 2)

select OrderID, ShipName from orders

[Function(Name="dbo.VariableResultShapes")]

[ResultType(typeof(VariableResultShapesResult1))]

[ResultType(typeof(VariableResultShapesResult2))]

public IMultipleResults VariableResultShapes([Parameter(DbType="Int")] System.Nullable<int> shape)

{

IExecuteResult result = this.ExecuteMethodCall(this, ((MethodInfo)(MethodInfo.GetCurrentMethod())), shape);

return ((IMultipleResults)(result.ReturnValue));

}

Northwnd db = new Northwnd(@"c:\northwnd.mdf");

// Assign the results of the procedure with an argument

// of (1) to local variable 'result'.

IMultipleResults result = db.VariableResultShapes(1);

// Iterate through the list and write results (the company names)

// to the console.

foreach(VariableResultShapesResult1 compName in

result.GetResult<VariableResultShapesResult1>())

{

Console.WriteLine(compName.CompanyName);

}

// Pause to view company names; press Enter to continue.

Console.ReadLine();

// Assign the results of the procedure with an argument

// of (2) to local variable 'result'.

IMultipleResults result2 = db.VariableResultShapes(2);

// Iterate through the list and write results (the order IDs)

// to the console.

foreach (VariableResultShapesResult2 ord in

result2.GetResult<VariableResultShapesResult2>())

{

Console.WriteLine(ord.OrderID);

}