Entity Framework Notes

www.enosislearning.com

Contents

[Introduction to Entity Framework 3](#_Toc518977872)

[Benefits 4](#_Toc518977873)

[Advantage of Entity Framework 4](#_Toc518977874)

[Difference between LINQ to SQL and EF is given below. 4](#_Toc518977875)

[Example of Entity Framework 5](#_Toc518977876)

[**Using the code** 5](#_Toc518977877)

[Step3 : Doing Insert operation 7](#_Toc518977878)

[Step4: Reading all the records 8](#_Toc518977879)

[Step5: Selecting a specific record 8](#_Toc518977880)

[Step6:Updating the record 8](#_Toc518977881)

[ENTITY FRAMEWORK TYPES OF QUERY 8](#_Toc518977882)

[1) LINQ to Entities: 9](#_Toc518977883)

[2) Entity SQL: 9](#_Toc518977884)

[3) Native SQL 10](#_Toc518977885)

[Stored Procedure Example using Entity Framework 10](#_Toc518977886)

[Importing Stored Procedure as Entity Function in Entity Framework 11](#_Toc518977887)

[EXAMPLE : TO DO CRUD OPERARIONS IN ENTITIY FRAMEWORK BY USING SP 15](#_Toc518977888)

[Stored Procedures using Entity- Framework 15](#_Toc518977889)

[Insert Stored Procedure 17](#_Toc518977890)

[Delete Stored Procedure 17](#_Toc518977891)

[Select Stored Procedure 17](#_Toc518977892)

[Update Stored Procedure 17](#_Toc518977893)

[ENTITY FRAMEWORKS SYNTAXES 26](#_Toc518977894)

[**When to use Single, SingleOrDefault, First and FirstOrDefault** 27](#_Toc518977895)

[**SELECTION USING ENTITY FRAMEWORK** 27](#_Toc518977896)

[**LINQ Method syntax:** 27](#_Toc518977897)

[**LINQ QUERY SYNTAX** 28](#_Toc518977898)

# Introduction to Entity Framework

Entity Framework is an Object Relational Mapper (ORM).

ADO.NET entity is an ORM (object relational mapping) which creates a higher abstract object model over ADO.NET components.

So rather than getting into dataset, datatables, command, and connection objects as shown in the below code, you work on higher level domain objects like customers, suppliers, etc.

It basically generates business objects and entities according to the database tables and provides the mechanism for:

1. Performing basic CRUD (Create, Read, Update, Delete) operations.
2. Easily managing "1 to 1", "1 to many", and "many to many" relationships.
3. Ability to have inheritance relationships between entities.
4. We can have all data access logic written in higher level languages.
5. The conceptual model can be represented in a better way by using relationships among entities.
6. The underlying data store can be replaced without much overhead since all data access logic is present at a higher level.

**Architecture of Entity Framework**



**Benefits**

The main and the only benefit of EF is it auto-generates code for the Model (middle layer), Data Access Layer, and mapping code, thus reducing a lot of development time.

### Difference between LINQ to SQL and EF is given below.

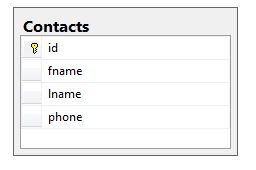
|  |  |
| --- | --- |
| **LINQ to SQL** | **Entity Framework** |
| **It only works with SQL Server Database.** | **It can works with various databases like Oracle, DB2, MYSQL, SQL Server etc.** |
| **It generates a .dbml to maintain the relation** | **It generates an .edmx files initially. The relation is maintained using 3 different files .csdl, .msl and .ssdl** |
| It has not support for complex type. | It has support for complex type. |
| It cannot generate database from model. | It can generate database from model. |
| It allows only one to one mapping between the entity classes and the relational tables /views. | It allows one-to-one, one-to-many & many-to-many mappings between the Entity classes and the relational tables /views |
| It allows you to query data using DataContext. | It allows you to query data using EntitySQL, ObjectContext, DbContext. |
| It provides a tightly coupled approach. | It provides a loosely coupled approach. Since its code first approach allows you to use Dependency Injection pattern which make it loosely coupled. |
| It can be used for rapid application development only with SQL Server. | It can be used for rapid application development with RDBMS like SQL Server, Oracle, DB2 and MySQL etc. |

# Example of Entity Framework

Entity Framework for performing simple CRUD operations.

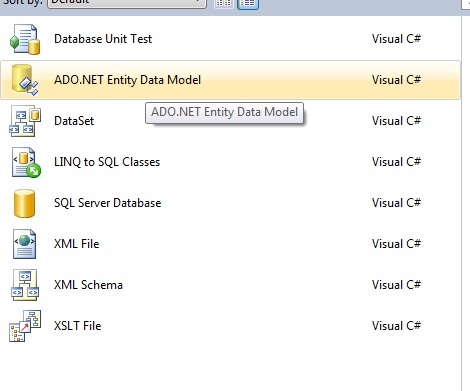
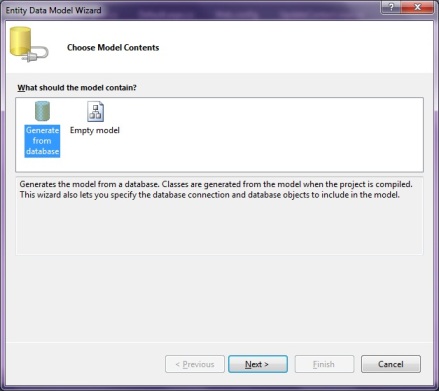
**Step1 : Creating the database**

Let's have a simple database with one table. Let's create a simple table for Contacts and we will perform CRUD operations on this table.



**Step2 : Adding the Entity Model to the Website**

Once we have the database ready, we can add the entity model to our website. We can do this by adding an *ADO.NET Entity Data Model* to the website.

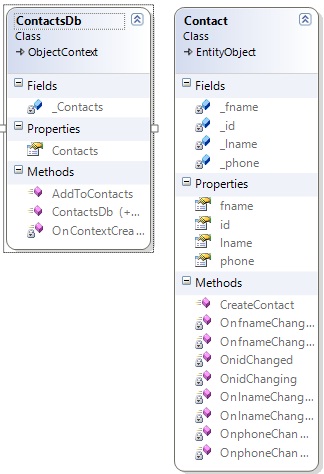
 

Once we select to add this data model to our website, we will have to select the approach we want to take for our Model's contents.

Once the Model is generated, the Entity for each table is generated. The generated entity for our contact table is:



Also, the classes for performing database operations are also created. We just need to know how to use these classes to perform database operations.



## Step3 : Doing Insert operation

|  |
| --- |
| Contact con = new Contact();  con.fname = TextBox1.Text;  con.lname = TextBox2.Text;  con.phone = TextBox3.Text;    ContactsDb db = new ContactsDb();  db.Contacts.Add(con);  db.SaveChanges(); |

## Step4: Reading all the records

|  |
| --- |
| We can retrieve the collection of Entities using the Model object to achieve this. The code snippet below will show how that can be done.  ContactsDb db = new ContactsDb();  Repeater1.DataSource = db.Contacts;  Repeater1.DataBind(); |

## Step5: Selecting a specific record

|  |
| --- |
| Let's say we want the functionality of updating/deleting a record on a single page then we will first have to select the record based on the ID, then update/delete the selected record.  Selection of any particular record (Contact) based on ID can be done as:  int idToupdate = Convert.ToInt32(Request.QueryString["id"].ToString());  ContactsDb db = new ContactsDb();  Contact con = db.Contacts.SingleOrDefault(p => p.id == idToupdate);  Once this code is executed, the Contact object will contain the required values. |

## Step6:Updating the record

|  |
| --- |
| If we want to update a record, then a simple update operation can be performed as:  int idToupdate = Convert.ToInt32(Request.QueryString["id"].ToString());  **ContactsDb db = new ContactsDb();**  **Contact con = db.Contacts.SingleOrDefault(p => p.id == idToupdate);**  **con.phone = TextBox1.Text;**  **db.SaveChanges();** |

# 

# ENTITY FRAMEWORK TYPES OF QUERY

You can query EDM mainly by three ways,

1) LINQ to Entities

2) Entity SQL

3) Native SQL

1) LINQ to Entities: L2E query syntax is easier to learn than Entity SQL. You can use your LINQ skills for querying with EDM. These are LINQ Method Syntax with Lamda expression and LINQ query syntax.

* 1. **LINQ Method syntax:**

//Querying with LINQ to Entities

using (var context = newSchoolDBEntities())

{

var L2EQuery = context.Students.where(s =>s.StudentName == “Bill”);

}

**1.2 LINQ Query syntax:**

using (var context = newSchoolDBEntities())

{

var L2EQuery = from st in context.Students

var studs = from st in db.students where st.StudentName == "Bill" select st;

}

First, you have to create an object of context class, which is SchoolDBEntities. You should initialize it in using() so that once it goes out of scope then it will automatically call Dispose() method of DbContext. In both the syntax above, context returns IQueryable.

2) Entity SQL: Entity SQL is another way to create a query. It is processed by the Entity Framework’s Object Services directly. **It returns ObjectQuery instead of IQueryable.**

You need ObjectContext to create a query using Entity SQL.

//Querying with Object Services and Entity SQL

String sqlString = "SELECT VALUE st FROM SchoolDBEntities.Students AS st WHERE st.StudentName == 'Bill'";

Var objctx = (ctx as IObjectContextAdapter).ObjectContext;

ObjectQuery<Student> student = objctx.CreateQuery<Student>(sqlString);

Student newStudent = student.First<Student>();

You can also use EntityConnection and EntityCommand to execute Entity SQL as shown below:

using (var con = new EntityConnection("name=SchoolDBEntities"))

{

con.Open();

EntityCommand cmd = con.CreateCommand();

cmd.CommandText = "SELECT VALUE st FROM SchoolDBEntities.Students as st where st.StudentName='Bill'";

Dictionary<int, string>dict = newDictionary<int, string>();

using (EntityDataReader rdr = cmd.ExecuteReader(CommandBehavior.SequentialAccess | CommandBehavior.CloseConnection))

{

while (rdr.Read())

{

int a = rdr.GetInt32(0);

var b = rdr.GetString(1);

dict.Add(a, b); }}}

3) Native SQL

You can execute native SQL queries for a relational database as shown below:

using (varctx = newSchoolDBEntities())

{

varstudentName = ctx.Students.SqlQuery("Select studentid, studentname, standardId from Student where studentname='Bill'").FirstOrDefault<Student>();

}

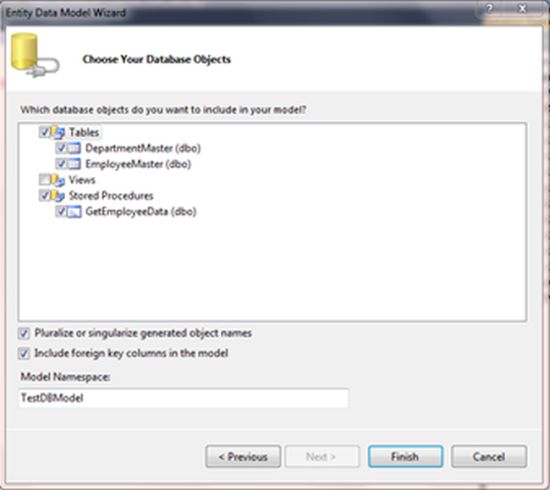
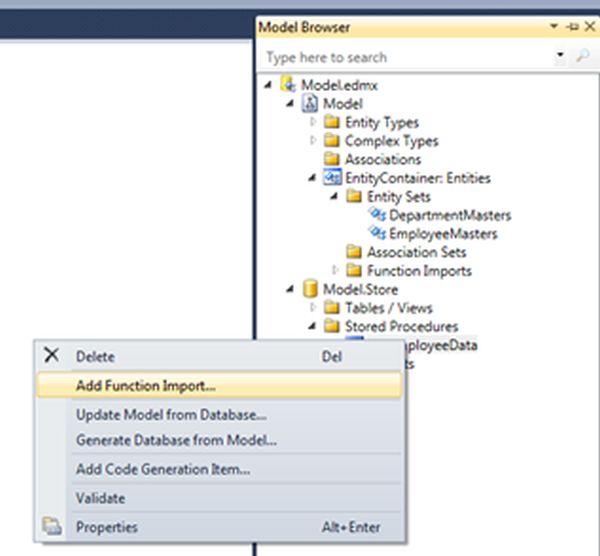
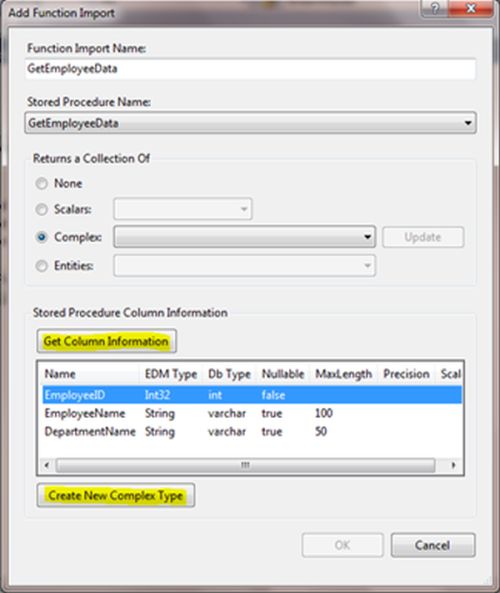
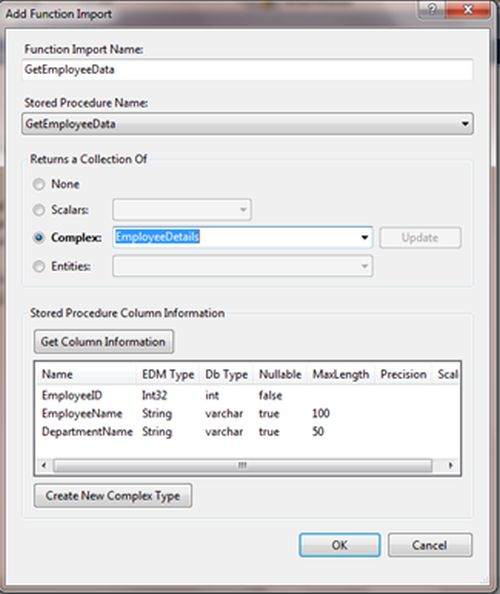
# Stored Procedure Example using Entity Framework

|  |
| --- |
| The following script help to generate the table with test data and a Stored Procedure. CREATE TABLE [dbo].[DepartmentMaster](                [DepartmentId] [int] IDENTITY(1,1) NOT NULL PRIMARY KEY,                [DepartmentName] [varchar](50) NULL,                [Status] [tinyint] NULL, )  CREATE TABLE [dbo].[EmployeeMaster](                [EmployeeID] [int] IDENTITY(1,1) NOT NULL PRIMARY KEY,                [EmployeeName] [varchar](100) NULL,                [DepartmentID] [int] NULL,                [Status] [tinyint] NULL  )  CREATE PROCEDURE [dbo].[GetEmployeeData] AS BEGIN                SELECT EmployeeID,EmployeeName,DepartmentName FROM EmployeeMaster E                                INNER JOIN DepartmentMaster D ON E.DepartmentID = D.DepartmentId END |

--Inserting some Dummy Data.

|  |
| --- |
| SET  IDENTITY\_INSERT [dbo].[DepartmentMaster] ON INSERT [dbo].[DepartmentMaster] ([DepartmentId], [DepartmentName], [Status]) VALUES (1, N'Maths', 0) INSERT [dbo].[DepartmentMaster] ([DepartmentId], [DepartmentName], [Status]) VALUES (2, N'English', 0) INSERT [dbo].[DepartmentMaster] ([DepartmentId], [DepartmentName], [Status]) VALUES (3, N'Physics', 0) SET IDENTITY\_INSERT [dbo].[DepartmentMaster] OFF  SET IDENTITY\_INSERT [dbo].[EmployeeMaster] ON INSERT [dbo].[EmployeeMaster] ([EmployeeID], [EmployeeName], [DepartmentID], [Status]) VALUES (1,N'Tejas', 1, 0) INSERT [dbo].[EmployeeMaster] ([EmployeeID], [EmployeeName], [DepartmentID], [Status]) VALUES (2,N'Rakesh', 1, 0) INSERT [dbo].[EmployeeMaster] ([EmployeeID], [EmployeeName], [DepartmentID], [Status]) VALUES (3,N'Jignesh', 2, 0) INSERT [dbo].[EmployeeMaster] ([EmployeeID], [EmployeeName], [DepartmentID], [Status]) VALUES (4,N'Kunal', 3, 0) SET IDENTITY\_INSERT [dbo].[EmployeeMaster] OFF |

Importing Stored Procedure as Entity Function in Entity Framework

The Entity Framework has the capability of importing a Stored Procedure as a function. We can also map the result of the function back to any entity type or complex type.  
  
The following is the procedure to import and use a Stored Procedure in Entity Framework.  
  
Step 1: Import Stored Procedure   
  
   
  
When we finish this process, the selected Stored Procedure is added to the model browser under the Stored Procedure Node.  
  
**Step 2:** Right-click Stored Procedure and select "Add Function Import".  
   
   
**Step 3:** Here, we can map a returned object of our Stored Procedure. The return type may be a scalar value or a collection of Model Entities or a collection of Complex (Custom) Entity. From this screen we can create a Complex Entity as well.  
  
    
   
  
  
Now, we can call the Stored Procedure as an entity function using the following code. The entity function returns a complex type called "EmployeeDetails".  
  
using (Entities context = new Entities())  
{  
    IEnumerable<EmployeeDetails>empDetails = context.GetEmployeeData();  
}  
  
2. Call Stored Procedure using ExecuteStoreQuery<T> function  
  
"ExecuteStoreQuery<T>" should be used to query data. This method only works if T has a Default Constructor and also a Property name is the same as the returned column names. "T" can be any generic class or any data type and it might not be a part of an EF generated entity.  
  
The following is the procedure to retrieve data using the "ExecuteStoreQuery<T>" method from a Stored Procedure.  
  
**Step 1:** The method "T" can be anything, it may be an EF Generated entity or it may be a Custom Entity, so first I am creating a Custom Entity "EmployeeDetail". Here the EmployeeDetail properties name must be the same as the returned column of the select statement of the Stored Procedure.

// Creating Custom class to hold result of Stored Procedure  
public class EmployeeDetail  
{  
    public int EmployeeID { get; set; }  
    public string EmployeeName { get; set; }  
    public string DepartmentName { get; set; }  
}

using (Entities context = new Entities())  
{  
        IEnumerable<EmployeeDetails>empDetails  =  context.ExecuteStoreQuery<EmployeeDetails>      
                                                                                            ("exec GetEmployeeData").ToList();  
}

// using DBContext (EF 4.1 and above)  
using (Entities context = new Entities())  
{  
        IEnumerable<EmployeeDetails>empDetails  =  context. Database.SqlQuery  
                                                                      < EmployeeDetails >("exec GetEmployeeData ", null).ToList();  
}  
  
3. Call Stored Procedure using DbDataReader  
  
We can also retrieve data or call a Stored Procedure using a SQL Connection Command and DbDataReader. The Object Context has a translate method that translates the entity data from DbDataReader into the requested type object. This method enables us to execute a standard ADO.Net query against a data source and return data rows into entity objects. Using the following code we can call a Stored Procedure and retrieve data in entity form.  
  
using (Entities context = new Entities())  
{  
  string ConnectionString = (context.Connection as EntityConnection).StoreConnection.ConnectionString;  
    SqlConnectionStringBuilder builder = new SqlConnectionStringBuilder(ConnectionString);  
    builder.ConnectTimeout = 2500;  
    SqlConnection con = new SqlConnection(builder.ConnectionString);  
    System.Data.Common.DbDataReader sqlReader;  
    con.Open();  
    using (SqlCommand cmd = con.CreateCommand())  
    {  
        cmd.CommandText = "GetEmployeeData";  
        cmd.CommandType = System.Data.CommandType.StoredProcedure;  
        cmd.CommandTimeout = 0;

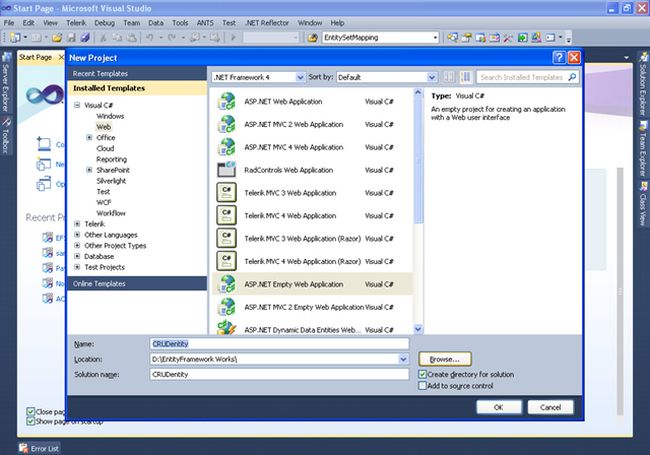
      sqlReader = (System.Data.Common.DbDataReader)cmd.ExecuteReader();  
      IEnumerable<EmployeeDetail>empDetails = context.Translate<EmployeeDetail>(sqlReader).ToList();  
    }  
}

# EXAMPLE : TO DO CRUD OPERARIONS IN ENTITIY FRAMEWORK BY USING SP

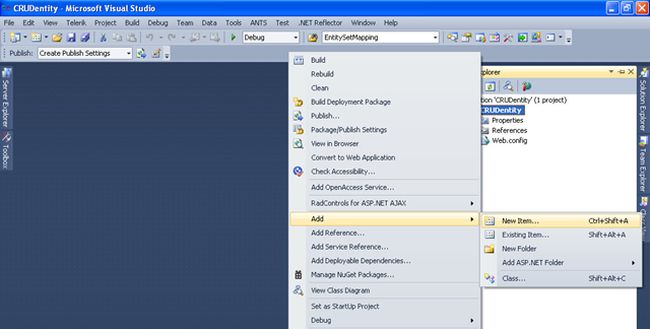
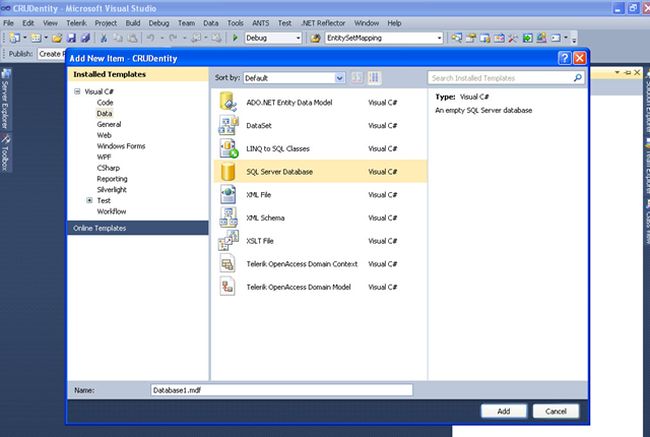
1. By calling Stored Procedures using ExecuteStoreCommand and ExecuteStoreQuery, without mapping to the Model.
2. By mapping Stored Procedures to the Model.

This example will show how we can map Stored Procedures to the Model to implement CRUD operations.

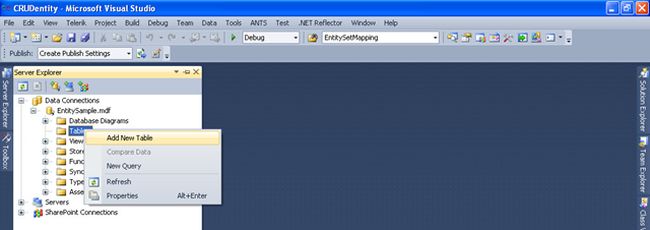
Create an empty web application from VS, see:



First let us create a sample table. Since I do not have SQL Server installed in my machine, I am adding a SQL Server database as in the following:

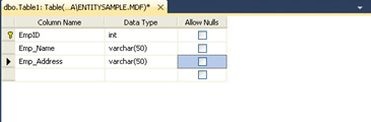
 

Here you can use the name *Database1.mdf* depending on your naming convention. Now in Server Explorer, you will see your database, we will add a table and some Stored Procedures as follows:



Add the required columns and save the table with the desired name. The most important aspect before you start working on Entity Framework is to have a Primary Key in your table.

Now my table looks as in the following, on which we are going to perform CRUD operations.



OK now let's create Stored Procedures for Insert, Update, Delete, and Select operations.

## Insert Stored Procedure

|  |
| --- |
| Create PROCEDURE dbo.InsertEmployee  (  @ID int,  @EmpNamevarchar(50),  @EmpAddressvarchar(50)  )  AS  Begin  insertinto Employee(EmpID,Emp\_Name,Emp\_Address)values(@ID,@EmpName,@EmpAddress)  END |

### Delete Stored Procedure

Create PROCEDURE dbo.deleteEmp

(

@ID int

)

As

Begin

Delete from Employee whereEmpID=@ID

End

### Select Stored Procedure

Create PROCEDURE dbo.SelectEmployee

As

Begin

select \* from Employee

End

### Update Stored Procedure

Create PROCEDURE dbo.UpdateEmployee

(

@ID int,

@EmpNamevarchar(50),

@EmpAddressvarchar(50)

)

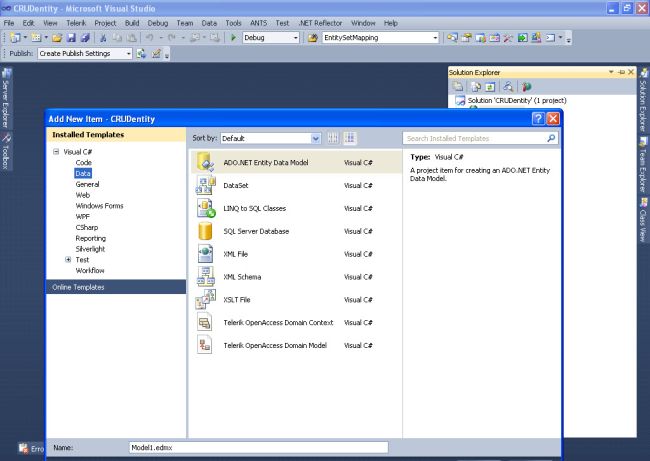
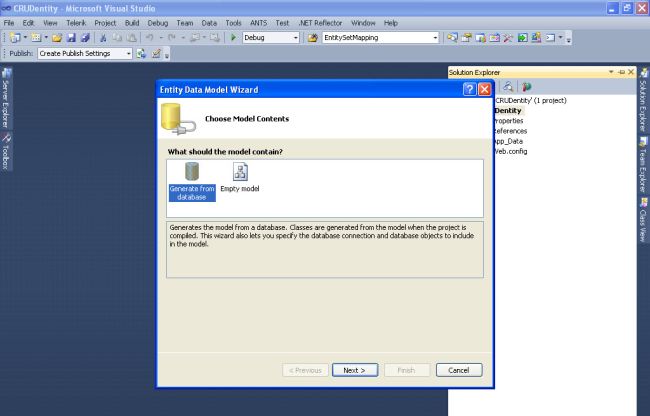
As

Begin

update Employee setEmp\_Name=@EmpName,Emp\_Address=@EmpAddresswhereEmpID=@ID

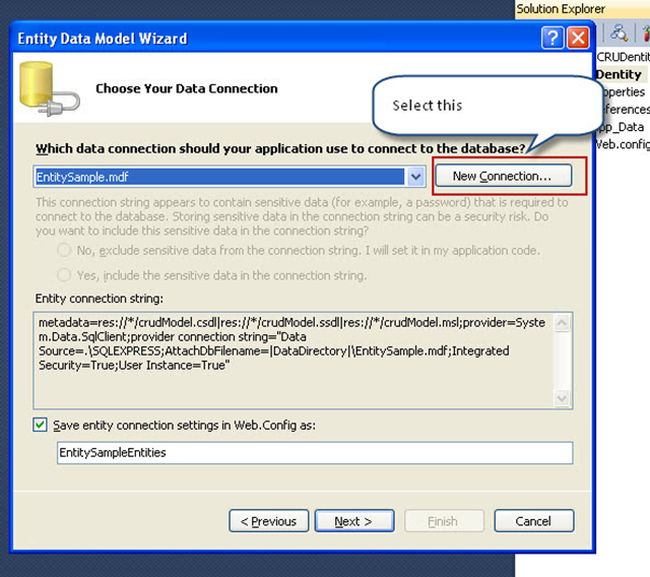
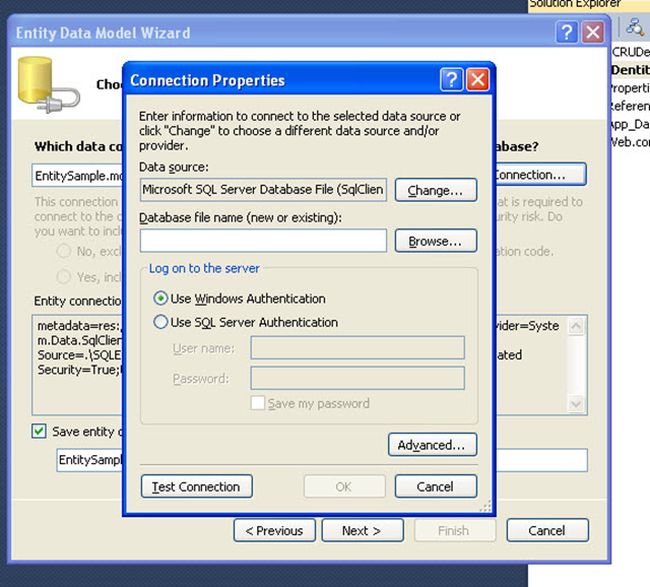
End

Adding an Entity Model to your application:

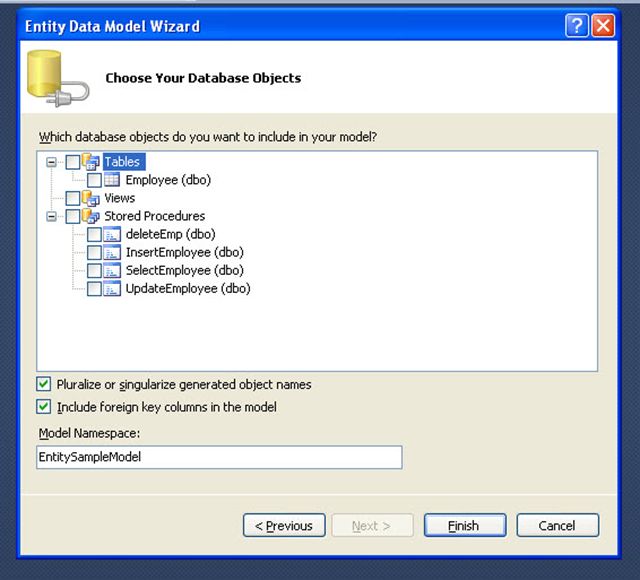
 

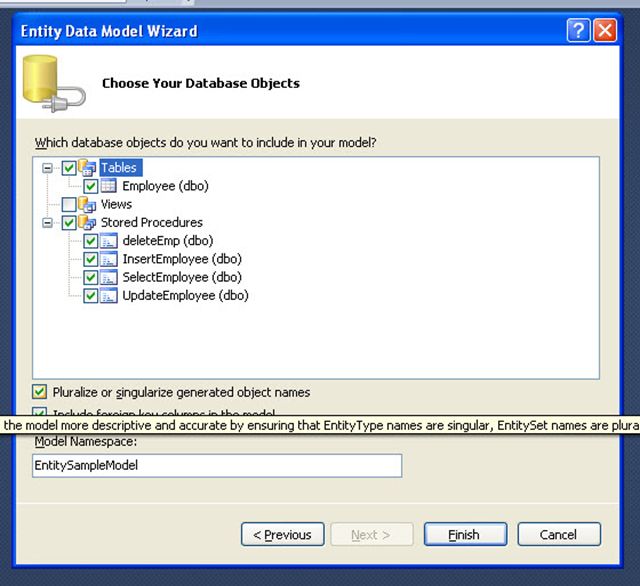
After adding a Model you will immediately have this Entity Data Model Wizard where you have to select Generate from the database and click on Next:

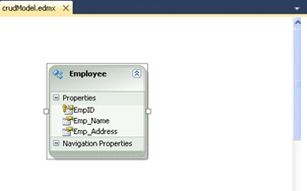
Select New Connection from Choose your data:

Click Test Connection and Next where you will find all your tables and Stored Procedures that you have created. Select the required one.





Click on "Finish" after you are finished, then you will see your model with the tables you added and if there are any relations it will also map them. 

Now we are finished with creating the database and adding it to an Entity Model. Now we will see how to perform CRUD operations without mapping the Stored Procedures to the model.

I also included some LINQ queries wherever needed, for example to auto-generate Employee ID and binding the drop-down list.

using System;

using System.Linq;

using System.Data.SqlClient;

namespace CRUDentity

{

publicpartialclass crud : System.Web.UI.Page

{

EntitySampleEntities entities = new EntitySampleEntities();

protectedvoid Page\_Load(object sender, EventArgs e)

{

if (!IsPostBack)

{

checkMax();

loadGrid();

bindDDL();

}

}

Protected void btnInsert\_Click(object sender, EventArgs e)

{

var ietsParameterID = new SqlParameter("@ID", System.Data.SqlDbType.Int);

ietsParameterID.Value = Convert.ToInt16(txtEmpID.Text);

var ietsParameterEmpName = new SqlParameter("@EmpName", txtEmployeeName.Text);

var ietsParameterEmpAddress = new SqlParameter("@EmpAddress", txtAddress.Text);

**entities.ExecuteStoreCommand("InsertEmployee @ID,@EmpName,@EmpAddress",**

**ietsParameterID, ietsParameterEmpName, ietsParameterEmpAddress);**

loadGrid();

checkMax();

bindDDL();

txtAddress.Text = string.Empty;

txtEmployeeName.Text = string.Empty;

}

publicvoid checkMax()

{

int? maxEmpID = entities.Employees.Max(q => (int?)q.EmpID);

if (maxEmpID != null)

{

maxEmpID = maxEmpID + 1;

txtEmpID.Text = maxEmpID.ToString();

}

else

{

maxEmpID = 1000;

txtEmpID.Text = maxEmpID.ToString();

}

}

Public void loadGrid()

{

var selectData = entities.ExecuteStoreQuery<Employee>("SelectEmployee").ToList();

grdEmployess.DataSource = selectData;

grdEmployess.DataBind();

}

publicvoid bindDDL()

{

var display = from e in entities.Employees select new { e.EmpID };

ddlEmpID.DataSource = display.ToList();

ddlEmpID.DataTextField = "EmpID";

ddlEmpID.DataValueField = "EmpID";

ddlEmpID.DataBind();

ddlEmpID.Items.Insert(0, "--Select--");

ddleditEmpID.DataSource = display.ToList();

ddleditEmpID.DataTextField = "EmpID";

ddleditEmpID.DataValueField = "EmpID";

ddleditEmpID.DataBind();

ddleditEmpID.Items.Insert(0, "--Select--");

}

protectedvoid btnDelete\_Click(object sender, EventArgs e)

{

if (ddlEmpID.SelectedItem.Text != "--Select--")

{

var ietsParameterID = new SqlParameter("@ID", ddlEmpID.SelectedItem.Text);

entities.ExecuteStoreCommand("deleteEmp @ID", ietsParameterID);

loadGrid();

checkMax();

bindDDL();

}

}

protectedvoid btnUpdate\_Click(object sender, EventArgs e)

{

if (ddleditEmpID.SelectedItem.Text != "--Select--")

{

var ietsParameterID = new SqlParameter("@ID", System.Data.SqlDbType.Int);

ietsParameterID.Value = Convert.ToInt16(ddleditEmpID.SelectedItem.Text);

var ietsParameterEmpName = new SqlParameter("@EmpName", txtedtEmployeeName.Text);

var ietsParameterEmpAddress = new SqlParameter("@EmpAddress", txtedtEmpAddress.Text);

entities.ExecuteStoreCommand("UpdateEmployee @ID,@EmpName,@EmpAddress",ietsParameterID, ietsParameterEmpName, ietsParameterEmpAddress);

loadGrid();

}

}

protectedvoid ddleditEmpID\_SelectedIndexChanged(object sender, EventArgs e)

{

if (ddleditEmpID.SelectedItem.Text != "--Select--")

{

int id = Convert.ToInt16(ddleditEmpID.SelectedValue.ToString());

var display = from e1 in entities.Employees

where e1.EmpID.Equals(id)

select new { e1.Emp\_Name, e1.Emp\_Address };

foreach (var v in display)

{

txtedtEmployeeName.Text = v.Emp\_Name;

txtedtEmpAddress.Text = v.Emp\_Address;

}

}

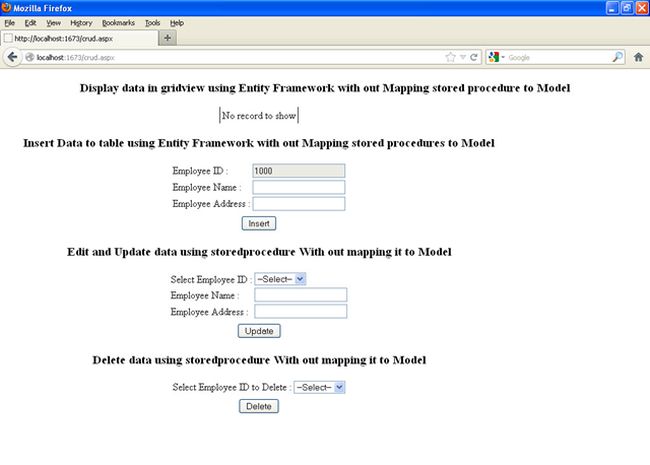
}

}

}

***Sample screenshots***

When you first run the application:



Framework without mapping Stored Procedures to the Model.

# ENTITY FRAMEWORKS SYNTAX

|  |  |  |
| --- | --- | --- |
| COMMAND | DESCRIPTION | SYNTAX |
| CREATE OBJECT | To Create a object of entity framework context class | ContactsDb db = new ContactsDb(); |
| Add / Insert | To add a new record into the table | db.Contacts.AddObject(con);  db.SaveChanges(); |
| Select/View | To select the records from the table | Repeater1.DataSource = db.Contacts;  Repeater1.DataBind(); |
| Delete | To delete the records from the table | USER u = db.USERS.Find(id);  db.USERS.Remove(u);  db.SaveChanges(); |
| Update | To update the records from the table | db.Entry(u).State = EntityState.Modified;  db.SaveChanges(); |
| Single | It returns a single specific element from a collection of elements if element match found. | //try to get single element 10 if exist  Console.WriteLine(data.Single(d => d == 10)); //result:10 |
| SingleorDefault | It returns a single specific element from a collection of elements if element match found. | data.SingleOrDefault(d => d == 100)  //result:0, since default value is 0  var student = (from st in context.Students where  st.StudentID == 1  select st).SingleOrDefault<Student>(); |
|  |  |  |
| Firstor  Default | It returns first specific element from a collection of elements if one or more than one match found for that element. A default value is returned, if no match is found for that element in the collection. | var student = (from st in ctx.Students where st.StudentName == "Student1" select st).FirstOrDefault<Student>(); |
| **When to use Single, SingleOrDefault, First and FirstOrDefault**  You should take care of following points while choosing Single, SingleOrDefault, First and FirstOrDefault:   1. When you want an exception to be thrown if the result set contains many records, use Single or SingleOrDefault. 2. When you want a default value is returned if the result set contains no record, use SingleOrDefault. 3. When you always want one record no matter what the result set contains, use First or FirstOrDefault. 4. When you want a default value if the result set contains no record, use FirstOrDefault. | | |
| **SELECTION USING ENTITY FRAMEWORK** | | |
| **LINQ Method syntax:**  var L2EQuery = context.Students.where(s =>s.StudentName == “Bill”); | | |
| **LINQ QUERY SYNTAX**  var studs = from st in db.students where st.StudentName == "Bill" select st; | | |
| //Querying with Object Services and Entity SQL  String sqlString = "SELECT VALUE st FROM SchoolDBEntities.Students " +  "AS st WHERE st.StudentName == 'Bill'";  Var objctx = (ctx as IObjectContextAdapter).ObjectContext;  ObjectQuery<Student> student = objctx.CreateQuery<Student>(sqlString);  StudentnewStudent = student.First<Student>(); | | |
|  | | |

|  |  |  |
| --- | --- | --- |
| COMMAND | DESCRIPTION | SYNTAX |
| **ExecuteStoreCommand** | This command is used to execute a stored procedure. | var ietsParameterID = new SqlParameter("@ID", System.Data.SqlDbType.Int);  ietsParameterID.Value = Convert.ToInt16(txtEmpID.Text);  var ietsParameterEmpName = new SqlParameter("@EmpName", txtEmployeeName.Text);  var ietsParameterEmpAddress = new SqlParameter("@EmpAddress", txtAddress.Text);  **entities.ExecuteStoreCommand("InsertEmployee @ID,@EmpName,@EmpAddress", ietsParameterID, ietsParameterEmpName, ietsParameterEmpAddress);** |
| ExecuteStoreQuery | Execute a query and returns the result | var selectData = entities.ExecuteStoreQuery<Employee>("Select \* from Employee").ToList();  grdEmployess.DataSource = selectData;  grdEmployess.DataBind(); |

|  |  |  |
| --- | --- | --- |
| SELECT | To select particular columns from the list | var query = context.Products  .Select(product =>new  {  ProductId = product.ProductID,  ProductName = product.Name  });  Console.WriteLine("Product Info:");  foreach (varproductInfoin query)  {  Console.WriteLine("Product Id: {0} Product name: {1} ",  productInfo.ProductId, productInfo.ProductName);  } |
| WHERE | To filter records | Example  Var onlineOrders = context.SalesOrderHeaders  .Where(order =>order.OnlineOrderFlag == true) |
| Order by | To sort the records in asc or desc | var student1 = from s in db.Students  order by s.StudentName  ascending select s; |
| JOIN | To Join Two Tables | var result = from emp in db.EMPLOYEEs join de in db.DEPARTMENTs on emp.DEPTID equals de.DEPTID  select new  {  EmployeeId = emp.EMPID,  EmployeeName = emp.NAME,  Department = de.DEPTNAME,  Location = de.LOCATION}; |