**What is Entity Framework?**

Writing and managing ADO.Net code for data access is a tedious and monotonous job. Microsoft has provided an O/RM framework called "Entity Framework" to automate database related activities for your application.

Microsoft has given the following definition of Entity Framework:

*The Microsoft ADO.NET Entity Framework is an Object/Relational Mapping (ORM) framework that enables developers to work with relational data as domain-specific objects, eliminating the need for most of the data access plumbing code that developers usually need to write. Using the Entity Framework, developers issue queries using LINQ, then retrieve and manipulate data as strongly typed objects. The Entity Framework's ORM implementation provides services like change tracking, identity resolution, lazy loading, and query translation so that developers can focus on their application-specific business logic rather than the data access fundamentals.*

Entity framework is an Object/Relational Mapping (O/RM) framework. It is an enhancement to ADO.NET that gives developers an automated mechanism for accessing & storing the data in the database.

Entity framework is useful in three scenarios. First, if you already have existing database or you want to design your database ahead of other parts of the application. Second, you want to focus on your domain classes and then create the database from your domain classes. Third, you want to design your database schema on the visual designer and then create the database and classes.

The following figure illustrates the above scenarios.

[](http://www.entityframeworktutorial.net/Images/EF-overview.png)

As per the above figure, EF creates data access classes for your existing database, so that you can use these classes to interact with the database instead of ADO.Net directly.

EF can also create the database from your domain classes, thus you can focus on your domain-driven design.

EF provides you a model designer where you can design your DB model and then EF creates database and classes based on your DB model.

**What is O/RM?**

ORM is a tool for storing data from domain objects to relational database like MS SQL Server, in an automated way, without much programming. O/RM includes three main parts: Domain class objects, Relational database objects and Mapping information on how domain objects map to relational database objects (tables, views & storedprocedures). ORM allows us to keep our database design separate from our domain class design. This makes the application maintainable and extendable. It also automates standard CRUD operation (Create, Read, Update & Delete) so that the developer doesn't need to write it manually.

A typical ORM tool generates classes for the database interaction for your application as shown below.

[](http://www.entityframeworktutorial.net/Images/ORM.png)

Visit wikipedia for more information on [Object-relational Mapping](http://en.wikipedia.org/wiki/Object-relational_mapping)

There are many ORM frameworks for .net in the market such as DataObjects.Net, NHibernate, OpenAccess, SubSonic etc. Entity Framework is an open source ORM framework from Microsoft.

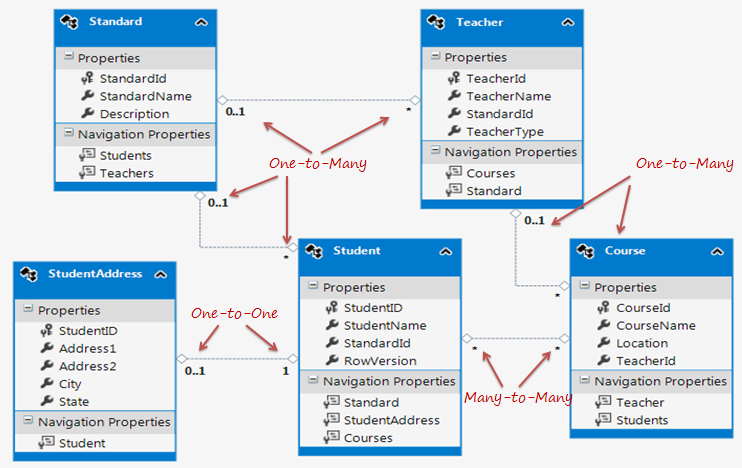
Please note that **Entity Framework is an open source framework** by Microsoft. You can contribute to the Entity Framework project on [codeplex](https://entityframework.codeplex.com/).

**Entity Relationships:**

Here, you will learn how entity framework manages the relationships between entities.

Entity framework supports three types of relationships, same as database: 1) One to One 2) One to Many, and 3) Many to Many.

We have created an Entity Data Model for the SchoolDB database in the [Create Entity Data Model](http://www.entityframeworktutorial.net/EntityFramework5/create-dbcontext-in-entity-framework5.aspx) section. The following figure shows the visual designer for that EDM with all the entities and relationships among them.

[](http://www.entityframeworktutorial.net/images/entityrelation2.png)

## One-to-One Relationship:

As you can see in the above figure, Student and StudentAddress have a One-to-One relationship (zero or one). A student can have only one or zero address. Entity framework adds Student navigation property into StudentAddress entity and StudentAddress navigation entity into Student entity. Also, StudentAddress entity has StudentId property as PrimaryKey which makes it a One-to-One relationship.

As you can see in the above code, Student entity class includes StudentAddress navigation property and StudentAddress includes Student navigation property with foreign key property StudentId. This way EF handles one-to-one relationship between entities.

## One-to-Many Relationship:

The Standard and Teacher entities have a One-to-Many relationship marked by multiplicity where 1 is for One and \* is for many. This means that Standard can have many Teachers whereas Teacher can associate with only one Standard.

To represent this, The Standard entity has the collection navigation property **Teachers** (please notice that it's plural), which indicates that one Standard can have a collection of Teachers (many teachers). And Teacher entity has a Standard navigation property (Not a Collection) which indicates that Teacher is associated with one Standard. Also, it contains StandardId foreign key (StandardId is a PK in Standard entity). This makes it One-to-Many relationship.

As you can see in the above code snippet, Standard entity class has Teachers property of type ICollection, so that it can contain multiple Teacher objects. (It initializes Teachers property with HashSet<Teacher> in the constructor, so that you can add Teacher objects into collection without worrying about initializing it.)

Also, Teacher entity class includes Standard property with StandardId for foreign key property. Entity framework includes this foreign key property because we checked **Include foreign key columns in the model** in the EDM wizard while creating EDM in the [Create Entity Data Model](http://www.entityframeworktutorial.net/EntityFramework5/create-dbcontext-in-entity-framework5.aspx) section.

## Many-to-Many Relationship:

Student and Course have Many-to-Many relationships marked by \* multiplicity. It means one Student can enrol for many Courses and also, one Course can be be taught to many Students.

The database design includes StudentCourse joining table which includes the primary key of both the tables (Student and Course table). Entity Framework represents many-to-many relationships by not having entityset for the joining table in CSDL, instead it manages this through mapping.

As you can see in the above figure, Student entity includes Courses property and Course entity includes Students property to represent many-to-many relationship between them.

**Note:** Entity framework supports many-to-many relationship only when the joining table (StudentCourse in this case) does **NOT** include any columns other than PKs of both the tables. If the join tables contain additional columns, such as DateCreated, then the EDM creates entity for middle table as well and you will have to manage CRUD operation for many-to-many entities manually.

# Entity Lifecycle:

Before we work on CRUD operation (Create, Read, Update, Delete), it's important to understand the entity lifecycle and how it is being managed by the EntityFramework.

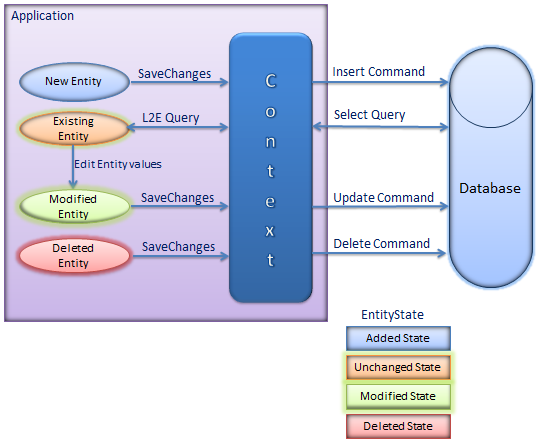
During an entity's lifetime, each entity has an entity state based on the operation performed on it via the context (DbContext). The entity state is an enum of type *System.Data.Entity.EntityState* that includes the following values:

1. Added
2. Deleted
3. Modified
4. Unchanged
5. Detached

The Context not only holds the reference to all the objects retrieved from the database but also it holds the entity states and maintains modifications made to the properties of the entity. This feature is known as *Change Tracking*.

The change in entity state from the Unchanged to the Modified state is the only state that's automatically handled by the context. All other changes must be made explicitly using proper methods of DbContext and DbSet.

The following figure illustrates how the operation performed on entity changes its' states which, in turn, affects database operation.

[](http://www.entityframeworktutorial.net/images/EF5/entity-states.png)

As you can see in the above figure, new entity in context has Added entity state. So the context will execute insert command to the database. In the same way, when you retrieve an existing entity using L2E queries, it will have Unchanged state, this is because you have just retrieved an entity and hasn't performed any operation on it yet. When you modify values of existing entity, it changes its state to Modified which in turn will execute update command on SaveChanges. Deleted entity from context will have Deleted state which in turn will execute delete command to the database.

So, in this way, operations performed on entities changes states. Context builds and executes database commands based on the state of an entity.

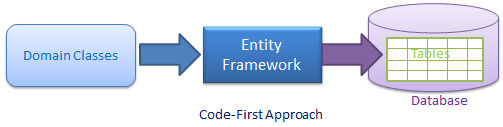
**Code First development with Entity Framework:**

Entity Framework supports three different development approaches to use entity framework in your application.

1. Code First
2. Model First
3. Database first

**Code First:** In the Code First approach, you avoid working with visual model designer (EDMX) completely. You write your POCO classes first and then create database from these POCO classes.

Developers who follow the path of Domain-Driven Design (DDD) principles, prefer to begin by coding their domain classes first and then generating the database required to persist their data.

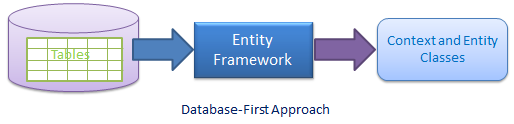
[](http://www.entityframeworktutorial.net/images/EF5/code-first.png)

# Model First development with Entity Framework:

**Model First:** In the Model First approach, you create Entities, relationships, and inheritance hierarchies directly on the design surface of EDMX and then generate database from your model.

# Database First development with Entity Framework:

We have seen this approach in [Create Entity Data Model](http://www.entityframeworktutorial.net/EntityFramework5/create-dbcontext-in-entity-framework5.aspx) where we created the EDM, context and entity classes from an existing database. So, when you generate EDMX from an existing database then it is a Database First approach.

[](http://www.entityframeworktutorial.net/images/EF5/databasefirst.png)

Entity Data Model can be updated whenever database schema changes. Also, database-first approach supports stored procedure, view, etc.

# Choose development approach with Entity Framework:

We have seen Code-first, Model-first and Database-first approaches in the previous sections. So, now you have to make a decision about which development approach to use in your application. The following figure illustrates the decision tree.

# Querying with EDM:

We have created EDM, DbContext, and entity classes in the previous sections. Here, you will learn the different types of queries an entity framework supports, which is in turn converted into SQL query for the underlaying database.

Entity framework supports three types of queries: 1) LINQ to Entities, 2) Entity SQL, and 3) Native SQL

**1) LINQ to Entities:** Language-Integrated Query (LINQ) is a powerful query language introduced in Visual Studio 2008. You can use LINQ in C# or Visual Basic to query different data sources. LINQ-to-Entities operates on entity framework entities to access the data from the underlying database. You can use LINQ method syntax or query syntax when querying with EDM. Visit [LINQ Tutorials](http://www.tutorialsteacher.com/linq/linq-tutorials) to learn LINQ step-by-step.

### LINQ Method syntax:

//Querying with LINQ to Entities

using (var context = new SchoolDBEntities())

{

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

var student = L2EQuery.FirstOrDefault<Student>();

}

### LINQ Query syntax:

using (var context = new SchoolDBEntities())

{

var L2EQuery = from st in context.Students

where st.StudentName == "Bill"

select st;

var student = L2EQuery.FirstOrDefault<Student>();

}

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 syntaxes above, context returns IQueryable.

Learn different types of LINQ to Entities projection query in the [Projection Query](http://www.entityframeworktutorial.net/querying-entity-graph-in-entity-framework.aspx) section.

**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.

The following code snippet shows the same query result as L2E query above.

//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 = new Dictionary<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);

}

}

}

EntityDataReader doesn't return ObjectQuery. Instead, it returns the data in rows & columns.

Visit [MSDN blog](http://blogs.msdn.com/b/adonet/archive/2007/05/30/entitysql.aspx) to learn Entity SQL.

**3) Native SQL:** You can execute native SQL queries for a relational database as shown below:

using (var ctx = new SchoolDBEntities())

{

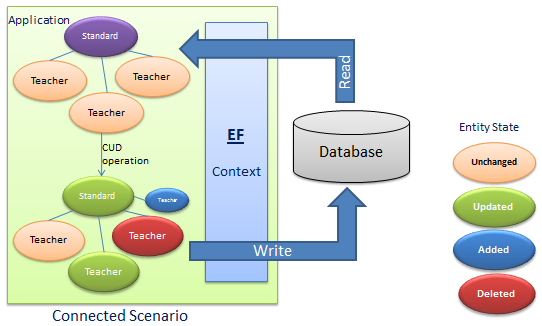
var studentName = ctx.Students.SqlQuery("Select studentid, studentname, standardId from Student where studentname='Bill'").FirstOrDefault<Student>();

}

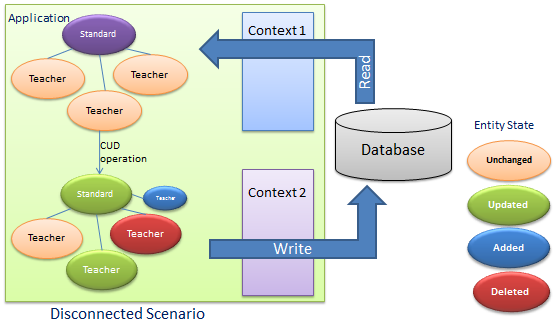
# Persistence in Entity Framework

There are two scenarios when persisting an entity using EntityFramework, connected and disconnected scenarios.

**Connected Scenario:** This is when an entity is retrieved from the database and persist is used in the same context. Context object doesn't destroy between entity retrieval and persistence of entities.

[](http://www.entityframeworktutorial.net/images/EF5/persistance-fg1.PNG)

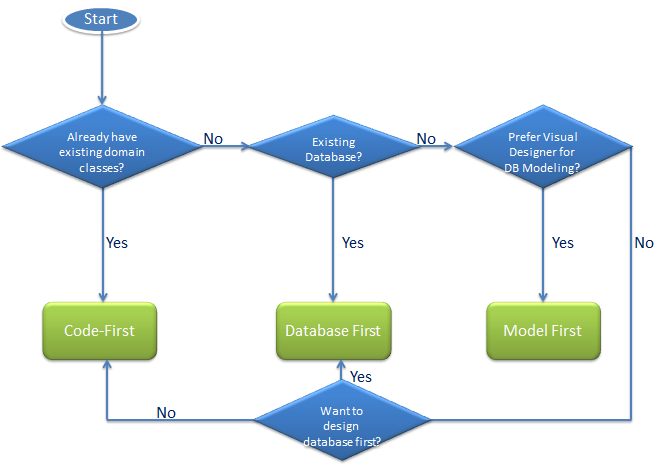
**Disconnected Scenario:** Disconnected scenario is when an entity is retrieved from the database and the changed entities are submitted using the different objects in the context. The following example illustrates disconnected scenario:

[](http://www.entityframeworktutorial.net/images/EF5/persistance-fg2.PNG)

As per the above scenario, Context1 is used for read operation and then Context1 is destroyed. Once entities change, the application submits entities using Context2 - a different context object.

Disconnected scenario is complex because the new context doesn't know anything about the modified entity so you will have to instruct the context of what has changed in the entity. In the figure below, the application retrieves an entity graph using Context 1 and then the application performs some CUD (Create, Update, Delete) operations on it and finally, it saves the entity graph using Context 2. Context 2 doesn't know what operation has been performed on the entity graph in this scenario.

Learn how to do the CRUD operation in the connected scenario in the next section.

[](http://www.entityframeworktutorial.net/images/EF5/choose-modeling.png)

As per the above figure, if you already have an existing application with domain classes, then you can use the code-first approach because you can create a database from your existing classes in this approach. If you have an existing database, then you can create an EDM from an existing database in the database-first approach. If you do not have an existing database or domain classes, and you prefer to design your DB model on the visual designer, then go for Model-first approach.