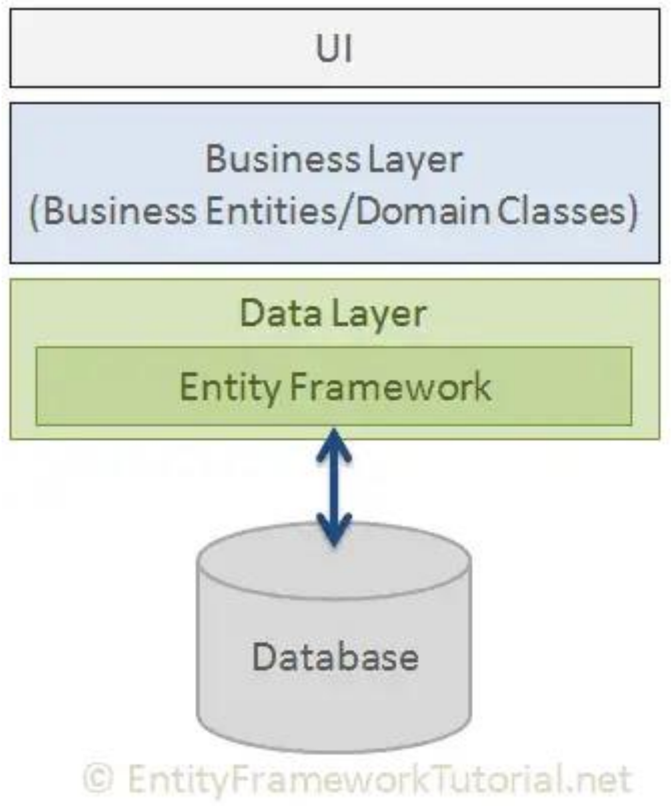
Module 2

Entity Framework

- Entity Framework is an Object Relational Mapper (ORM) which is a type of tool that simplifies mapping between objects in your software to the tables and columns of a relational database.
- An ORM takes care of creating database connections and executing commands, as well as taking query results and automatically materializing those results as your application objects.
- An ORM also helps to keep track of changes to those objects, and when instructed, it will also persist those changes back to the database for you.

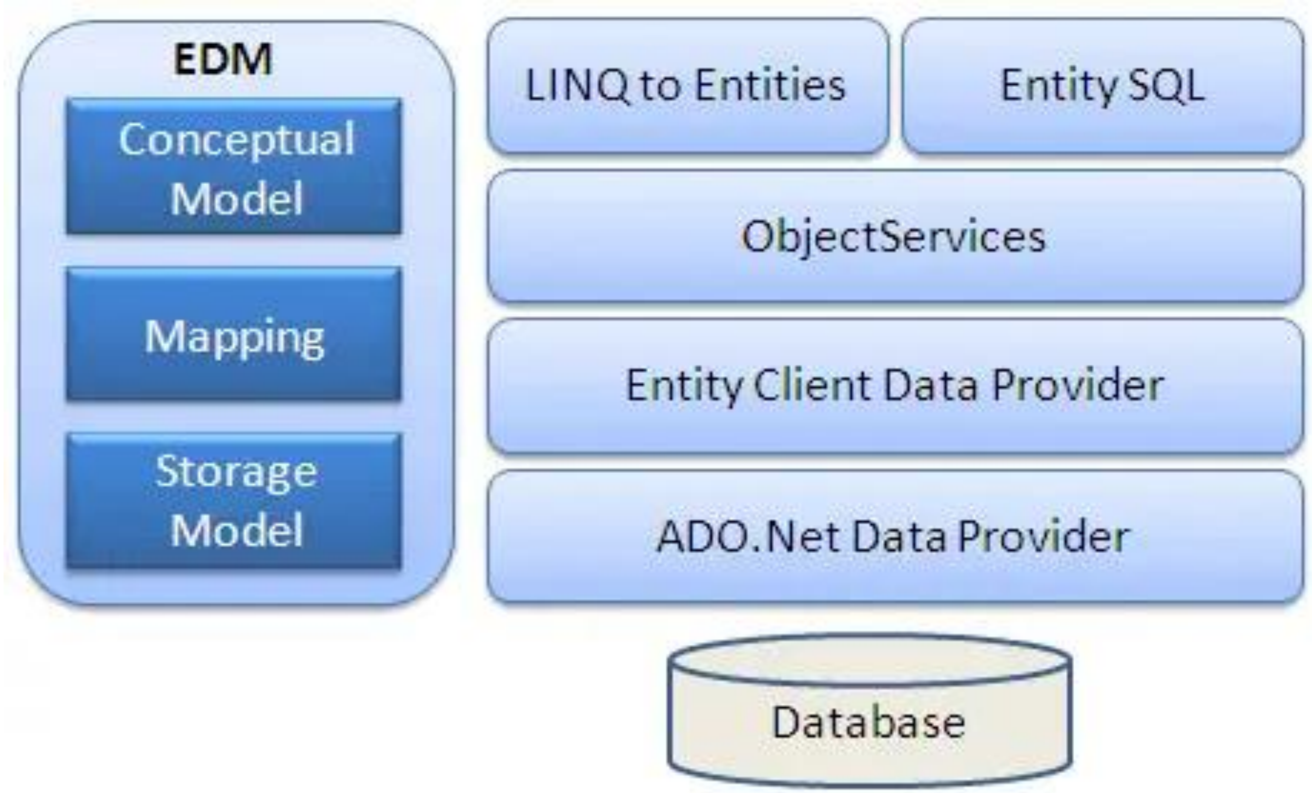
Entity Framework


UI
Business Layer
(Business Entities/Domain Classes)
Data Layer
Entity Framework
Database
© EntityFrameworkTutorial.net

Entity Framework

- Cross-platform: EF Core is a cross-platform framework which can run on Windows, Linux and Mac.
- Modelling: EF (Entity Framework) creates an EDM (Entity Data Model) based on POCO (Plain Old CLR Object) entities with get/set properties of different data types. It uses this model when querying or saving entity data to the underlying database.
- Querying: EF allows us to use LINQ queries (C#/VB.NET) to retrieve data from the underlying database. The database provider will translate this LINQ queries to the database-specific query language (e.g. SQL for a relational database). EF also allows us to execute raw SQL queries directly to the database.
- Change Tracking: EF keeps track of changes occurred to instances of your entities (Property values) which need to be submitted to the database.

- Saving: EF executes INSERT, UPDATE, and DELETE commands to the database based on the changes occurred to your entities when you call the SaveChanges() method. EF also provides the asynchronous SaveChangesAsync() method.


EDM
Conceptual Model
Mapping
Storage Model
LINQ to Entities
Entity SQL
ObjectServices
Entity Client Data Provider
ADO.Net Data Provider
Database

- EDM (Entity Data Model): EDM consists of three main parts - Conceptual model, Mapping and Storage model.
- Conceptual Model: The conceptual model contains the model classes and their relationships. This will be independent from your database table design.
- Storage Model: The storage model is the database design model which includes tables, views, stored procedures, and their relationships and keys.
- Mapping: Mapping consists of information about how the conceptual model is mapped to the storage model.

- LINQ to Entities: LINQ-to-Entities (L2E) is a query language used to write queries against the object model. It returns entities, which are defined in the conceptual model. You can use your LINQ skills here.
- Entity SQL: Entity SQL is another query language (For EF 6 only) just like LINQ to Entities. However, it is a little more difficult than L2E and the developer will have to learn it separately.
- Object Service: Object service is a main entry point for accessing data from the database and returning it back. Object service is responsible for materialization, which is the process of converting data returned from an entity client data provider (next layer) to an entity object structure.

- Entity Client Data Provider: The main responsibility of this layer is to convert LINQ-to-Entities or Entity SQL queries into a SQL query which is understood by the underlying database. It communicates with the ADO.Net data provider which in turn sends or retrieves data from the database.
- ADO.Net Data Provider: This layer communicates with the database using standard ADO.Net.

Development Approaches with Entity Framework

- Database-First
- Code-First
- Model-First

Code-First Approach

- Use this approach when you do not have an existing database for your application. In the code-first approach, you start writing your entities (domain classes) and context class first and then create the database from these classes using migration commands.

Example

- Let's assume that we want to create a simple application for XYZ School.
- Users of this School application should be able to add and update students, grades, teachers, and courses information.
- Instead of designing database tables first, let's start creating classes for our school domain, as and when needed.
- First, create the Student and Grade classes where every Student is associated with one Grade as shown below.

- DbContext corresponds to your database (or a collection of tables and views in your database) whereas a DbSet corresponds to a table or view in your database.
- You will be using a DbContext object to get access to your tables and views (which will be represented by DbSet's) and you will be using your DbSet's to get access, create, update, delete and modify your table dat

Context Class in Entity Framework

- It represent a session with the underlying database using which you can perform CRUD (Create, Read, Update, Delete) operations.

EF Core Stored Procedure

- A stored procedure is a **group of one or more pre-compiled SQL statements** into a logical unit.

EF Core Stored Procedure

```
CREATE PROCEDURE [schema_name].procedure_name
          @parameter_name data_type,
          ....
          parameter_name data_type
AS
  BEGIN
    -- SQL statements
    -- SELECT, INSERT, UPDATE, or DELETE statement
  END
```

EF Core Stored Procedure

- **EXEC** procedure_name;

```
CREATE PROCEDURE studentList
AS
BEGIN
   SELECT name, age, salary
   FROM STUDENT
   ORDER BY salary;
END;
```

ADO.NET

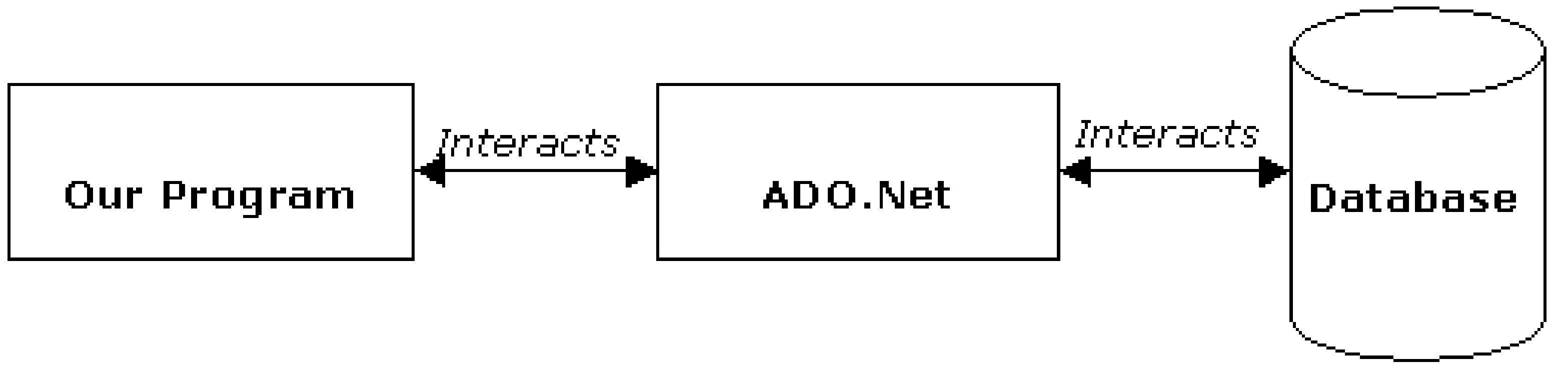
.NET Data Access and Manipulation

What is ADO.NET?

- A data-access technology that enables applications to connect to data stores and manipulate data contained in them in various ways
- Former version was ADO (ActiveX Data Object)

What is ADO.NET?

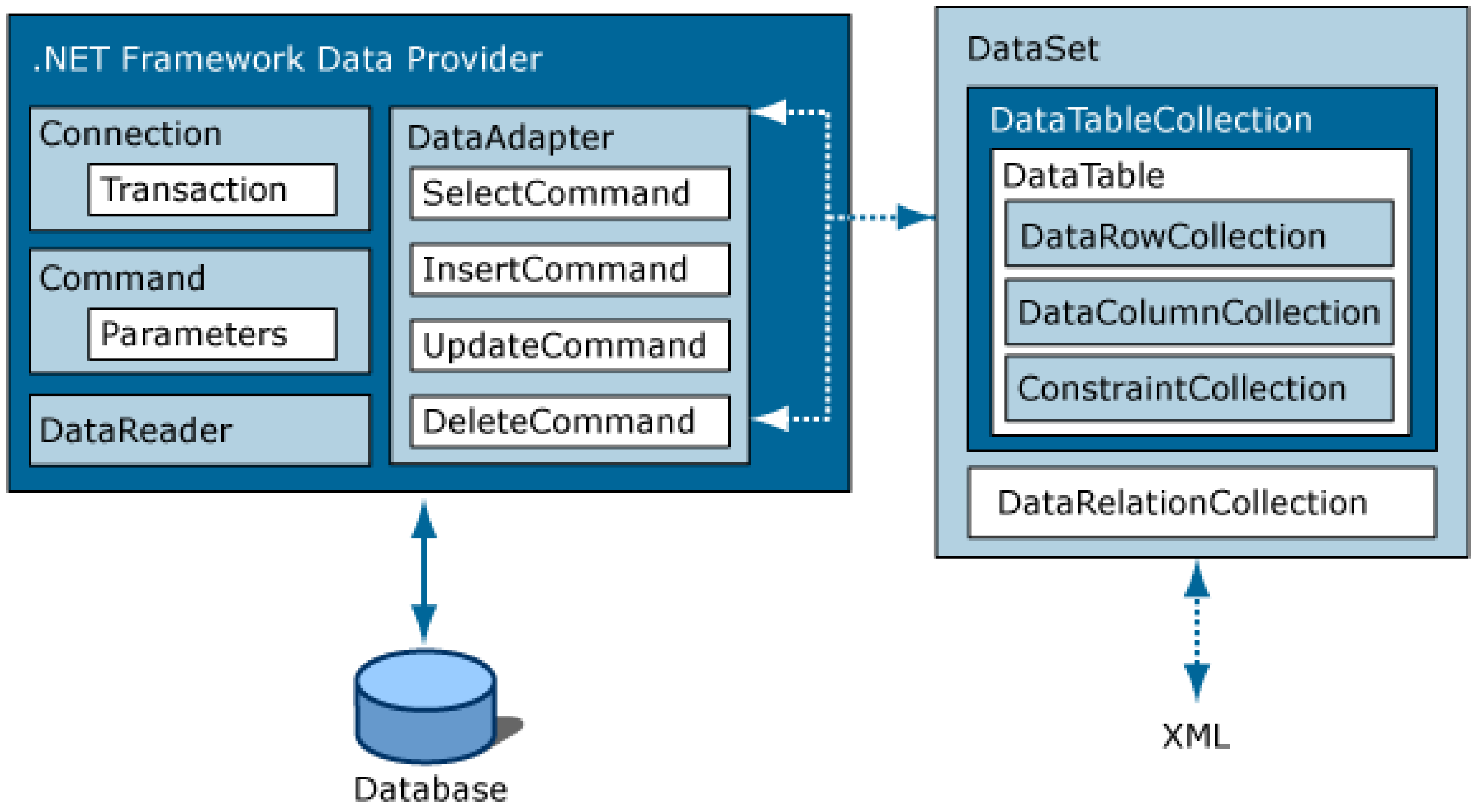
- An object oriented framework that allows you to interact with database systems



Objective of ADO.NET

- Support disconnected data architecture,
- Tight integration with XML,
- Common data representation
- Ability to combine data from multiple and varied data sources
- Optimized facilities for interacting with a database

ADO.NET Architecture



ADO.NET Core Objects

- Core namespace: System.Data
- .NET Framework data providers:

Data Provider	**Namespace**
SQL Server	`System.Data.SqlClient`
OLE DB	`System.Data.OleDb`
ODBC	`System.Data.Odbc`
Oracle	`System.Data.OracleClient`

ADO.NET Core Objects

Object	**Description**
`Connection`	Establishes a connection to a specific data source. (Base class: DbConnection)
`Command`	Executes a command against a data source. Exposes **Parameters** and can execute within the scope of a **Transaction** from a **Connection**. (The base class: DbCommand)
`DataReader`	Reads a forward-only, read-only stream of data from a data source. (Base class: DbDataReader)

Steps of Data Acces : Connected Environment

- Create connection
- Create command (select-insert-update-delete)
- Open connection
- If SELECT -> use a **`DataReader`** to fetch data
- If UPDATE,DELETE, INSERT -> use command object's methods
- Close connection

```
static void Main()
{
    string connectionString =
                      Properties.Settings.Default.connStr;
    string queryString = "SELECT CategoryID, CategoryName FROM
                                                dbo.Categories;";
    SqlConnection connection = new
                                 SqlConnection(connectionString);

    SqlCommand command = new SqlCommand(queryString,connection);
    try
    {
        connection.Open();
        SqlDataReader reader = command.ExecuteReader();
        while (reader.Read())
        {
            Console.WriteLine("\t{0}\t{1}",reader[0],reader[1]);
        }
        reader.Close();
         connection.close();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}
```

EXAMPLE

Connected – Update, Delete, Insert

- Command class core methods:
    - `ExecuteNonQuery` : Executes a SQL statement against a connection object
    - `ExecuteReader`: Executes the CommandText against the Connection and returns a `DbDataReader`
    - `ExecuteScalar`: Executes the query and returns the first column of the first row in the result set returned by the query

Connected – Update, Delete, Insert

```
string connString =
       Properties.Settings.Default.connStr;
SqlConnection conn = new
        SqlConnection(connString);
SqlCommand cmd = new SqlCommand("delete from
    Customers" + "where custID=12344", conn);
conn.Open();
cmd.ExecuteNonQuery();
conn.Close();
```

Can be an update or insert command