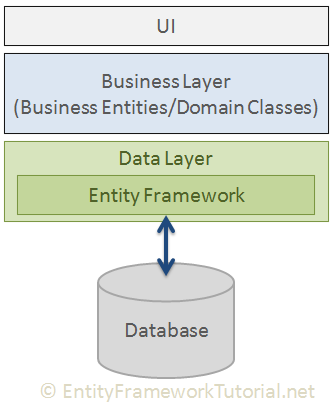
What is Entity Framework?

Prior to .NET 3.5, we (developers) often used to write ADO.NET code or Enterprise Data Access Block to save or retrieve application data from the underlying database. We used to open a connection to the database, create a DataSet to fetch or submit the data to the database, convert data from the DataSet to .NET objects or vice-versa to apply business rules. This was a cumbersome and error prone process. Microsoft has provided a framework called "Entity Framework" to automate all these database related activities for your application.

* Entity Framework is an open-source [ORM framework](https://en.wikipedia.org/wiki/Object-relational_mapping) for .NET applications supported by Microsoft.
* It enables developers to work with data using objects of domain specific classes without focusing on the underlying database tables and columns where this data is stored.
* With the Entity Framework, developers can work at a higher level of abstraction when they deal with data, and can create and maintain data-oriented applications with less code compared with traditional applications.

Official Definition: **“Entity Framework is an object-relational mapper (O/RM) that enables .NET developers to work with a database using .NET objects. It eliminates the need for most of the data-access code that developers usually need to write.”**

The following figure illustrates where the Entity Framework fits into your application.

[](https://www.entityframeworktutorial.net/images/basics/ef-in-app-architecture.png)

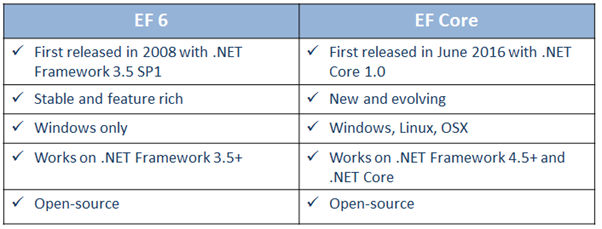
As per the above figure, Entity Framework fits between the business entities (domain classes) and the database. It saves data stored in the properties of business entities and also retrieves data from the database and converts it to business entities objects automatically.

Entity Framework Features

* **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.
* **Concurrency:** EF uses Optimistic Concurrency by default to protect overwriting changes made by another user since data was fetched from the database.
* **Transactions:** EF performs automatic transaction management while querying or saving data. It also provides options to customize transaction management.
* **Caching:** EF includes first level of caching out of the box. So, repeated querying will return data from the cache instead of hitting the database.
* **Built-in Conventions:** EF follows conventions over the configuration programming pattern, and includes a set of default rules which automatically configure the EF model.
* **Configurations:** EF allows us to configure the EF model by using data annotation attributes or Fluent API to override default conventions.
* **Migrations:** EF provides a set of migration commands that can be executed on the NuGet Package Manager Console or the Command Line Interface to create or manage underlying database Schema.

Entity Framework Latest Versions

Microsoft introduced Entity Framework in 2008 with .NET Framework 3.5. Since then, it released many versions of Entity Framework. Currently, there are two latest versions of Entity Framework: EF 6 and EF Core. The following table lists important difference between EF 6 and EF Core.

[](https://www.entityframeworktutorial.net/images/basics/ef6-vs-efcore.png)

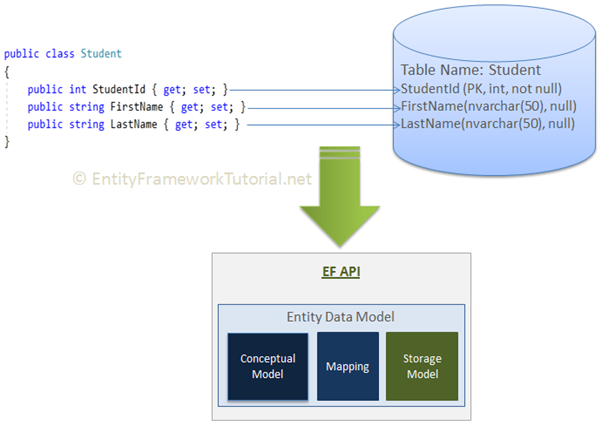
EF 6 Version History

| EF Version | Release Year | .NET Framework |
| --- | --- | --- |
| EF 6 | 2013 | .NET 4.0 & .NET 4.5, VS 2012 |
| EF 5 | 2012 | .NET 4.0, VS 2012 |
| EF 4.3 | 2011 | .NET 4.0, VS 2012 |
| EF 4.0 | 2010 | .NET 4.0, VS 2010 |
| EF 1.0 (or 3.5) | 2008 | .NET 3.5 SP1, VS 2008 |

# How Entity Framework Works?

Entity Data Model

The very first task of EF API is to build an Entity Data Model (EDM). EDM is an in-memory representation of the entire metadata: conceptual model, storage model, and mapping between them.

[](https://www.entityframeworktutorial.net/images/basics/ef-edm.png)

**Conceptual Model:** EF builds the conceptual model from your domain classes, context class, default conventions followed in your domain classes, and configurations.

**Storage Model:** EF builds the storage model for the underlying database schema. In the code-first approach, this will be inferred from the conceptual model. In the database-first approach, this will be inferred from the targeted database.

**Mappings:** EF includes mapping information on how the conceptual model maps to the database schema (storage model).

EF performs CRUD operations using this EDM. It uses EDM in building SQL queries from LINQ queries, building INSERT, UPDATE, and DELETE commands, and transform database result into entity objects.

# EF performs CRUD operations using this EDM. It uses EDM in building SQL queries from LINQ queries, building INSERT, UPDATE, and DELETE commands, and transform database result into entity objects.

Querying

EF API translates LINQ-to-Entities queries to SQL queries for relational databases using EDM and also converts results back to entity objects.

# https://www.entityframeworktutorial.net/images/basics/ef-querying.png

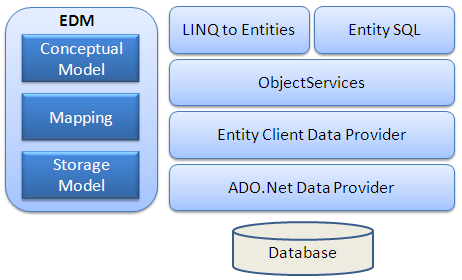
Saving

EF API infers INSERT, UPDATE, and DELETE commands based on the state of entities when the SaveChanges() method is called. The ChangeTrack keeps track of the states of each entity as and when an action is performed.

# https://www.entityframeworktutorial.net/images/basics/ef-saving.png

# Entity Framework Architecture

he following figure shows the overall architecture of the Entity Framework.

[](https://www.entityframeworktutorial.net/Images/ef-architecture.PNG)

Let's look at the components of the architecture individually.

**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.

Context Class in Entity Framework

The context class is a most important class while working with EF 6 or EF Core. It represent a session with the underlying database using which you can perform CRUD (Create, Read, Update, Delete) operations.

The context class in Entity Framework is a class which derives from [System.Data.Entity.DbContext](https://docs.microsoft.com/en-us/dotnet/api/system.data.entity.dbcontext" \t "_blank) in EF 6 and EF Core both. An instance of the context class represents Unit Of Work and Repository patterns wherein it can combine multiple changes under a single database transaction.

The context class is used to query or save data to the database. It is also used to configure domain classes, database related mappings, change tracking settings, caching, transaction etc.

The following SchoolContext class is an example of a context class.

using System.Data.Entity;

public class SchoolContext : DbContext

{

public SchoolContext()

{

}

// Entities

public DbSet<Student> Students { get; set; }

public DbSet<StudentAddress> StudentAddresses { get; set; }

public DbSet<Grade> Grades { get; set; }

}

In the above example, the SchoolContext class is derived from DbContext, which makes it a context class. It also includes an entity set for Student, StudentAddress, and Grade entities (learn about it next).

EntityState in Entity Framework

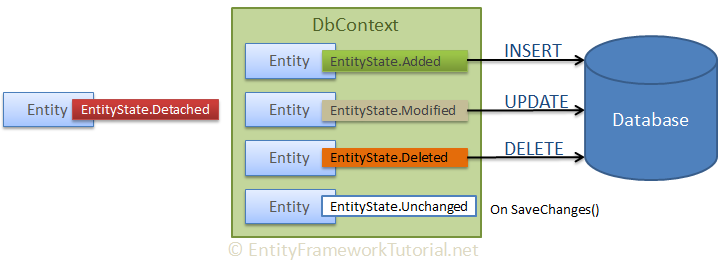
EF API maintains the state of each entity during an its lifetime. Each entity has a state based on the operation performed on it via the context class. The entity state represented by an enum System.Data.Entity.EntityState EF 6 and Microsoft.EntityFrameworkCore.EntityState in EF Core with the following values:

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

The Context not only holds the reference to all the entity objects as soon as retrieved from the database, but also keeps track of 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 or DbSet. (You will learn about these methods in EF 6 and EF Core sections.)

EF API builds and executes the INSERT, UPDATE, and DELETE commands based on the state of an entity when the context.SaveChanges() method is called. It executes the INSERT command for the entities with Added state, the UPDATE command for the entities with Modified state and the DELETE command for the entities in Deleted state. The context does not track entities in the Detached state. The following figure illustrates the significance of entity states:

[](https://www.entityframeworktutorial.net/images/basics/entity-states.png)

Thus, entity states play an important role in Entity Framework.

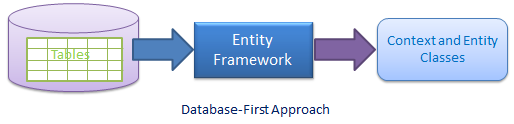
Development Approaches with Entity Framework

There are three different approaches you can use while developing your application using Entity Framework:

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

Database-First Approach

In the database-first development approach, you generate the context and entities for the existing database using EDM wizard integrated in Visual Studio or executing EF commands.

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

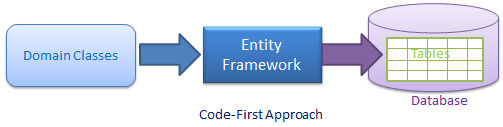
EF 6 supports the database-first approach extensively. Visit EF 6 DB-First section to learn about the database-first approach using EF 6.

EF Core includes limited support for this approach.

Code-First Approach

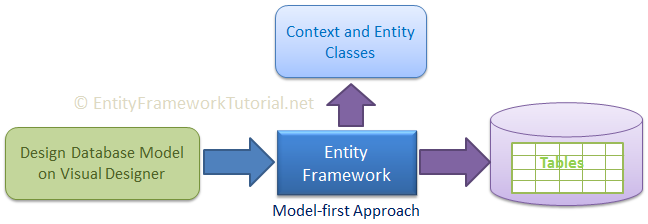
Use this approach when you do not have an existing database for your application. In the code-first approach, you start writing your entities (domain classes) and context class first and then create the database from these classes using migration commands.

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

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

Model-First Approach

In the model-first approach, you create entities, relationships, and inheritance hierarchies directly on the visual designer integrated in Visual Studio and then generate entities, the context class, and the database script from your visual model.

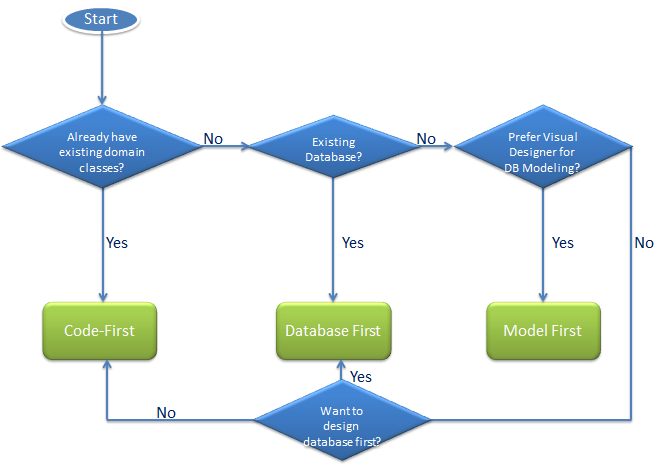
[](https://www.entityframeworktutorial.net/images/basics/model-first.png)

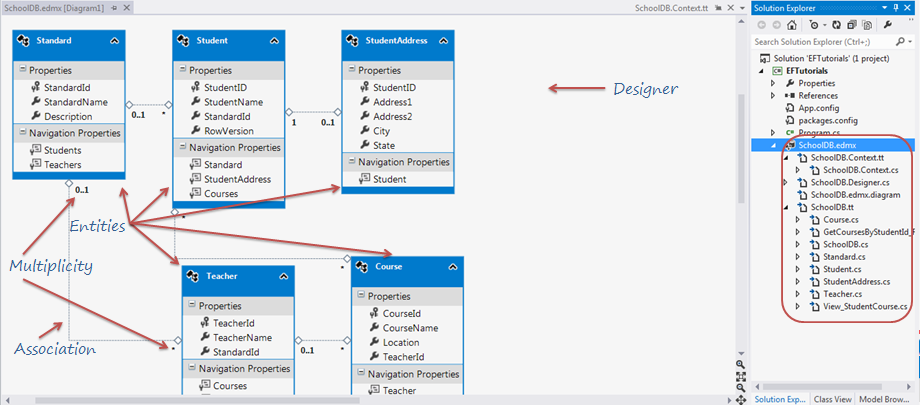
EF 6 includes limited support for this approach.

EF Core does not support this approach.

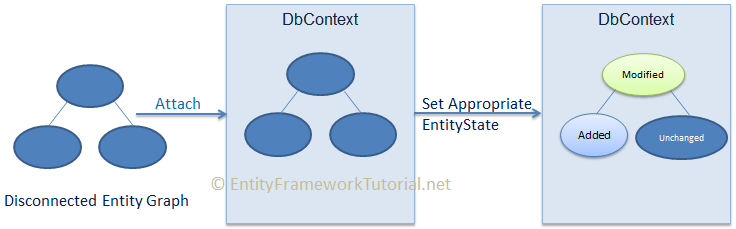
Choosing the Development Approach for Your Application

Use the following flow chart to decide which is the right approach to develop your application using Entity Framework:

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



Methods to Attach Disconnected Entities in EF 6



|  |  |
| --- | --- |
| **Parent Entity State** | **Entity State of child entities** |
| Added | Added |
| Modified | Unchanged |
| Deleted | All child entities will be null |

* DbContext.Entry()
* DbSet.Add()
* DbSet.Attach()

**Add or Remove Multiple Entities in Entity Framework**

IList<Student> newStudents = new List<Student>() {

new Student() { StudentName = "Steve" };

new Student() { StudentName = "Bill" };

new Student() { StudentName = "James" };

};

using (var context = new SchoolDBEntities())

{

context.Students.AddRange(newStudents);

context.SaveChanges();

}