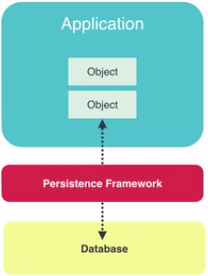
# **Entity Framework**

## **Introduction**

**It is an open-source ORM (Object Relational Mapping) Framework for the .NET applications supported by Microsoft.**

**It enables the developers to work with the data using the objects of domain-specific classes without focussing on the database tables and columns where the data is stored.**

**A persistence framework is middleware that assists in the storage and retrieval of information between applications and databases, especially relational databases.**

* It acts as a layer of abstraction for persisted data, bridging conceptual and technical differences between storage and utilization.

**With persistence framework, we will be able to load our objects from or saved them to a database.**

* We can write this persistence framework ourselves using plain ADO.NET classes like ***SqlConnection, SqlCommand, and SqlDataReader.***
* We need to write a lot of stored procedures read the data using ADO.NET object and manually Map our data with tables and records to the domain objects.

**It is a persistence framework that free us from to write stored procedures files.**

* No longer to write store procedures, manage database connections or manually mapped the database tables and records our domain objects.

**ORM** refers to the concept of Object Relational Mapping, that is the act of mapping records in a database (which may come from tables or views for example) to their object representation in an application (entity) or collections of entities together with their relationships.

**Persistence Frameworks** refer to frameworks that persist (store) data, normally into a database. Note that a persistence framework may persist to anything in reality, depending on the framework, even to an XML file for example. It is an abstraction layer between the database and the entities in an application.

Sometimes these terms are used interchangeably. Note that good persistence frameworks have their own rules of how to extract data, how to persist it, how to deal with what is called the impedance mismatch (look it up in Wiki), how to manage stale or dirty data in a predefined way, how to load data and related data, and so on and so forth.

### **Entity Framework Features**

**Cross-Platform**: Entity Framework Core is a cross-platform framework that can run on Windows, Linux, and Mac.

**Modelling**: Entity Framework (EF) creates an EDM (Entity Data Model), which is based on POCO (Plain Old CLR Object) entities with get/set properties of different data types. It uses this model when we have to query and to save the entity data to the underlying database.

**Querying**: Entity Framework allows us to use the LINQ Queries to retrieve the data from the database. The database provider will translate these LINQ queries to the database-specific query language (e.g., SQL for a relational database). Entity Framework also allows us to execute raw SQL queries directly to the database.

**Change Tracking**: Entity Framework keeps track of the changes that occurred to instances of the entities (Property Values) which needs to be submitted to the database.

**Saving**: Entity Framework executes the INSERT, UPDATE, and DELETE commands to the database based on the changes that occurred to the entities when we call the "***SaveChanges****()*" method. Entity Framework also provides the asynchronous "***SaveChangesAsync()"*** method.

**Concurrency**: Entity Framework uses Optimistic Concurrency by default to protect overwriting changes made by another user when data was fetched from the database.

**Transactions**: Entity Framework automates the management of the transaction while querying or saving the data. It also provides the options to customize the transaction management.

**Caching**: Entity Framework includes the first level of caching out of the box. So, repeated querying will return the data from the cache instead of hitting the database.

**Built-in Conventions**: Entity Framework follows the conventions over the configuration programming pattern, and includes a set of default rules which automatically configure the Entity Framework model.

**Configuration**: Entity Framework allows us to configure the Entity Framework model by using the data annotation attribute or Fluent API to override the default conventions.

**Migrations**: Entity Framework provides a set of the migration commands which can be executed on the ***NuGet*** Package Manager Console or the command-line interface to create or manage the database schema.

**Disadvantages of Entity Framework**

The disadvantages of EF are given bellow:

* Lazy loading is the main drawbacks of EF
* Its syntax is complicated
* Its logical schema is not able to understand business entities and relation among each other
* Logical schema of database is not capable of using certain parts of application
* It is not available for every RDMS
* Need to handle data in non-traditional way
* It does not work if we change any schema of the database. We need to update the schema on the solution.
* It is not good for huge domain model.

## **Packages**

**Some popular packages related to Entity Framework along with their descriptions:**

**EntityFramework**

**This is the *core* package for Entity Framework, providing the main functionality for working with databases using the Entity Framework ORM (Object-Relational Mapping).**

* It includes features such as entity CRUD operations, query execution, and change tracking.

**EntityFrameworkCore**

**It is a lightweight and cross-platform version of Entity Framework.**

* It is designed to work with .NET Core and provides a simplified and more efficient way to work with databases.
* It includes support for different database providers and enables you to build modern, scalable applications.

#### **Benefits**

1. **Object-Relational Mapping (ORM):** It enables you to work with databases using object-oriented code.

* It maps your database tables to classes (entities) in your code, allowing you to manipulate data as objects rather than writing complex SQL queries manually.

1. **Database Abstraction**: The package provides a high-level abstraction layer that allows you to interact with different database systems, such as SQL Server, MySQL, PostgreSQL, and SQLite, using a consistent API.

* It abstracts the underlying database details, allowing you to write database-agnostic code that can work with multiple database providers.

1. **Entity Modeling**: It includes a modeling framework that allows you to define the structure and relationships of your database entities using classes and annotations.

* It helps you create a conceptual model of your database schema and automatically generates the necessary database tables and relationships.

1. **Data Querying**: It provides powerful querying capabilities, allowing you to retrieve data from the database using LINQ (Language Integrated Query) expressions or method-based syntax.

* You can perform complex queries, apply filters, perform joins, and project data into custom result sets.

1. **Change Tracking and Persistence**: The package tracks changes made to your entities and generates the necessary SQL statements to persist those changes back to the database.

* It simplifies the process of inserting, updating, and deleting data, saving you from writing repetitive CRUD (Create, Read, Update, Delete) operations manually.

1. **Database Migrations**: It includes a migration framework that helps you manage database schema changes over time. With migrations, you can evolve your database schema as your application evolves, keeping the database in sync with your entity model.

* It handles the generation and execution of SQL scripts to apply the necessary schema changes.

1. **Cross-Platform Support**: It is designed to be cross-platform and can run on different operating systems, including Windows, macOS, and Linux.

* It enables you to develop database-driven applications using .NET Core or .NET 5+ on various platforms.

**EntityFramework.SqlServer**

**It is an extension package for Entity Framework that allows you to work specifically with Microsoft SQL Server databases.**

* It provides additional functionality and features tailored for SQL Server database operations within your Entity Framework Core project.
* It includes support for features specific to SQL Server, such as SQL Server-specific data types, functions, and optimizations.

#### **Benefits**

1. **Database Connectivity**: It enables your Entity Framework Core application to establish connections and communicate with Microsoft SQL Server databases.

* It provides the necessary components and configurations to connect to SQL Server databases seamlessly.

1. **SQL Server-Specific Features**: The package includes SQL Server-specific features and optimizations that enhance the interaction between your application and the database.

* This includes support for SQL Server-specific data types, functions, and query optimizations.

1. **Performance Improvements**: It incorporates optimizations specifically designed for SQL Server, which can improve the performance of data access operations.

* It leverages SQL Server's capabilities to execute queries efficiently and optimize database interactions.

1. **Migration Support**: The package facilitates the management of database migrations when using SQL Server as the target database.

* It includes commands and tools that allow you to create, apply, and revert migrations to evolve the database schema over time as your application evolves.

1. **Query Execution**: It optimizes the execution of Entity Framework queries against SQL Server databases.

* It takes advantage of SQL Server-specific query execution plans and optimizations to improve query performance and efficiency.

1. **Advanced Functionality**: The package offers additional features specific to SQL Server, such as support for stored procedures, table-valued functions, and database-specific operations.

* It allows you to leverage SQL Server's advanced capabilities within your Entity Framework Core application.

**EntityFramework.Design**

It **is used in Entity Framework Core projects to provide tools and functionality for design-time operations.**

* It helps with tasks like generating code from your entity models, creating and managing database migrations, and scaffolding entity models from an existing database.
* It provides ***design-time support*** for Entity Framework Core projects.
* It includes tools and components that assist with ***code generation*** and ***model discovery during development.***
* It enables commands such as ***migrations, scaffolding, and reverse engineering***.
* It is primarily used during development to generate code based on the entity models or to scaffold entity models from an existing database schema.

**EntityFramework.Tools**

**It is used in Entity Framework projects to provide command-line tools that assist in various development tasks.**

* These tools enable you to perform operations such as database migrations, generating code from an existing database, and scaffolding Entity Framework models from an existing database schema.
* It package provides command-line tools for Entity Framework Core projects.
* It includes commands such as migrations, database updates, and database creation.
* It is primarily used for runtime operations, such as applying migrations to the database or updating the database schema.

**Install the EntityFramework.Tools package**

**dotnet add package EntityFramework.Tools**

**You can also use the Entity Framework tools to scaffold entity models from an existing database schema.**

* For example, to generate the models based on an existing SQL Server database.

**dotnet ef dbcontext scaffold *"Server=(localdb)\MSSQLLocalDB; Database=MyDatabase;Trusted\_Connection=True;"* Microsoft.EntityFrameworkCore.SqlServer -o Models**

**EntityFramework.Proxies: This package enables lazy loading and change tracking proxies in Entity Framework. Lazy loading allows related entities to be loaded on-demand when accessed, while change tracking proxies enable change detection for entities.**

**EntityFramework.Relational: This package provides relational database-specific functionality for Entity Framework, including features like table-valued functions, database views, and more advanced query capabilities.**

**EntityFramework.InMemory: This package is useful for testing and development scenarios as it allows you to work with an in-memory database instead of a physical database. It provides a lightweight database provider that stores data in memory, making it fast and efficient for unit testing or prototyping.**

**EntityFramework.Extensions: This package offers additional extensions and utilities for Entity Framework, such as bulk operations for efficiently performing insert, update, and delete operations on a large number of entities.**

## **Workflows**

**There are three workflows to build a domain model using any Entity Framework.**

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

### **Database First**

* **We design our tables using visual designer and then have any EF to generate the domain classes based on the database.**
* **This is the traditional approach that a lot of developers have been doing for years.**

### **Code First**

* We create our domain classes and then EF generate the database tables for us.
* This approach was introduced in more recent years and it’s still new to many developer.

### **Model First**

* We use a UML diagram in visual studio to do model our classes and their associations and then based on this diagram EF generates that domain classes and database for us.
* Not recommended to use this approach due to the visual designer for modelling classes is very poor and hardly any developer uses it.

## **Demo Database-First Workflow**

**We create a simple database and a table then we have EF to generate a domain class based on the table.**

**Here’s the tricky part of the database first approach.**

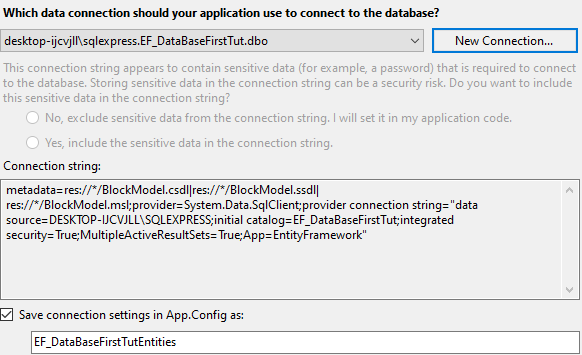
* We have this database here locally but in order to bring a different database like a test database or production database to this version we need a mechanism.
* There are tools out there that can compare the databases and bring them to the same version however sometimes they are not reliable and can causes issue due to dependency between tables and records.
* Manual approach is more reliable and flexible to create a change script every time we make a change to our database.
* We can store these database change scripts in the repository and them on any database to bring them to the current version.

### **Install EF package using Package manager console**

**Install-package Entity-Framework**

### **Add ADO.NET Entity Data Model**

This going to be conceptual model that represents the mapping between the database tables and domain classes.

* Go ***Solution Explorer > (Right Click) project >Add > new item > ADO.NET Entity Data Model***
* Select - ***EF designer from database*** (because we have database)
* Fill the database related data in dialog box and next.
* Select the required table, store procedures, etc. and NEXT.
* We get a security warning about running this template can potentially harm a computer. (Ignore and click on OK)
* Now ***ADO.NET Entity* Data *Model*** *which is stored in a file with* ***DMX*** *extension.*

**There are two important file within the *ADO.NET Entity Data Model* file.**

* **ModelName.Context,tt**
* .tt stand for T4 template. It’s a way to generate code based on a template.
* Within it (*ModelName.Context.cs*), we can see the actual generated code.
* Here a C# class that EF generated for me and this class derives from **DbContext class. (**The context is a class that is abstraction over the database**)**
* We have a property to *return type DBSet called Post.*
* **ModelName.tt**
* On expending the file in solution Explorer, we can see the Post class.
* Contain all the properties for Post.

**Whenever we make a change in my model.**

* **We need to start with database.**
* **We need to do change in datamodel.*edmx*** and refresh it.

**The DbContext is a class that is abstraction over the database**

* It provides a simple API to load the data from or save it to the database.

**DBSet** represent the tables in the database.

### **Use DBContext to work with a database.**

This can done with both approach in same way.

// create a instance of DbContext object

// DBContext class name - EF\_DataBaseFirstTutEntities1

EF\_DataBaseFirstTutEntities1 dbcontext = new EF\_DataBaseFirstTutEntities1();

// create a new post

var post = new master\_post()

{

Body = "Test Description",

DatePublished = DateTime.Now,

Title = "Post1",

PostId = 1

};

// add new post in db using dbContext instance

// added to dbset

dbcontext.master\_post.Add(post);

//push the data into db

dbcontext.SaveChanges();

## **Database-First or Code-First**

**Anything we want to do is possible with both code-first and database-first approach.**

**There are some misconceptions like**

* **Database-First gives us more control over database.**
* With Code-First also gives full control over the database.
* Code-First is for new projects.
* We can reverse engineer our existing database to create a Code-First Model then use code-First migrations for any subsequent changes afterward.

### **Some advantages of Code-First**

* **Full versioning of database.**
* We can bring our database to any version with running only one command.
* It is extremely helpful if we are maintaining different versions of an application.
* **Productivity**
* **It’s much faster to write code than use the mouse and a table designer.**

# **Code-First**

## **Demo Code-First Workflow**

**With code –first migration, we can take any database to any version, whether that version is higher or lower.**

### **Install EF package using Package manager console**

**Install-package Entity-Framework**

### **Create a model class**

**Create a model class with name Post and define all properties which are required and available in the database.**

class Post

{

[Key]

public int PostId { get; set; }

public DateTime DatePublished { get; set; }

public string Title { get; set; }

public string Body { get; set; }

}

### **Create a DBContext inheriting class**

// Use to connect with database for load and save data.

class ApplicationDbContext : DbContext

{

// create properties

public DbSet<Models.Post> Posts { get; set; }

}

### **Specifiy the connection string in web.config/ Appsetting.json (core)**

<connectionStrings>

<add name="ApplicationDbContext" connectionString="data source= DESKTOP-IJCVJLL\SQLEXPRESS; initial catalog=EFCodeFirstTut; integrated security=SSPI;" providerName="System.Data.SqlClient"/>

</connectionStrings>

### **Enable Migration**

// enable-**migration** for code-first (run only first time)

enable-migrations

**//add migration**

Add-migration DB\_Tbl-PostCreation

**// push migration to db**

Update-database -verbose

# **Building a Model using Code First Workflow.**

## **Introduction**

Some developer assumes that code first is only for ***Greenfield*** (New) projects where we don’t have an existing where we don’t have an existing database.

* So we are going to use Code-First for both existing or in new database.

## **Code First With New Database**

### **Create a Project and Install the Entity Framework.**

Open the Package Manager Console and Run Command to install the Entity Framework.

// to install the latest package

install-package EntityFramework

// to install the latest package

Install-Package EntityFramework -Version <number>

// Installing the Latest Preview

Install-Package EntityFramework -Pre

### **Create Model classes**

public class Courses

{

public int Id { get; set; }

public string Title { get; set; }

public string Description { get; set; }

public CourseLevel level { get; set; }

public float FullPrice { get; set; }

public Author author { get; set; }

public IList<Tag> Tags { get; set; }

}

public class Author

{

public int Id { get; set; }

public string Name { get; set; }

public IList<Courses> courses { get; set; }

}

public class Tag

{

public int Id { get; set; }

public string Name { get; set; }

public IList<Courses> courses { get; set; }

}

public enum CourseLevel

{

Beginner =1,

Intermediate =2,

Advanced = 3

}

### **Create DBContext**

It is a simple abstraction or the database that hides all the complexity around working with the connection’s commands, reading data and so on.

public class ApplicationDbContext : DbContext

{

// Constructor

public ApplicationDbContext() : base("name=DefaultConnection ")

{

// call this constructor to set connection string for db session

}

// create dbset

public DbSet<Course> Courses { get; set; }

public DbSet<Author> Authors { get; set; }

public DbSet<Tag> Tags { get; set; }

}

### **ConnectionString**

<connectionStrings>

<add name="ApplicationDbContext" connectionString="data source= DESKTOP-IJCVJLL\SQLEXPRESS; initial catalog=EFCodeFirstTut; integrated security=SSPI;" providerName="System.Data.SqlClient"/>

</connectionStrings>

Migration

// enable-**migration** for code-first (run only first time)

enable-migrations

**//add migration**

Add-migration DB\_Tbl-PostCreation

**// push migration to db**

Update-database -verbose

## **Code-First with an existing Database**

**We build a code first model using an existing database once the model is generated.**

* Once the model is created, we use the Code-First workflow for any subsequent changes.
* So instead of changing the database first and then updating your model.
* We make changes to the model using code and then use migration to bring the database up-to-date.

### **Benefits**

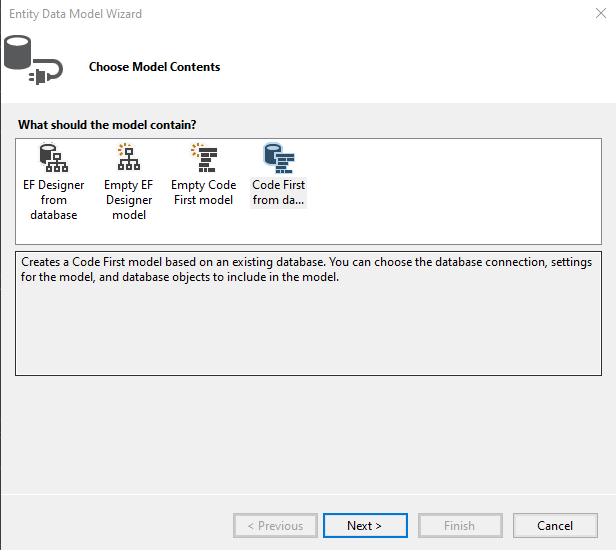
#### **Full versioning of database**

The moment we switch to code-first workflow, we get full versioning of our database.

#### **Less time wasted in designers**

We just write code which we do.

### **Practical**

1. Right click project> Add new item > select entity data model > select Code-First from database.