**Entity Framework:**

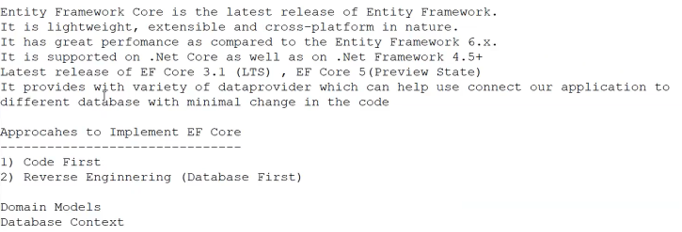
Data Access Mechanism which sits on the top of ADO.NET.

ORM: Object Relational Mapping framework. Does the mapping between database tables and .NET classes.

Supports connecting to different databases like MS SQL Server, Oracle etc.

Supports 1:1, 1:M and M:M relationships.

**Entity Faremwork Core:**



**Code First:** Model Classes and Database Context class are written and based on that Database and Tables are generated.

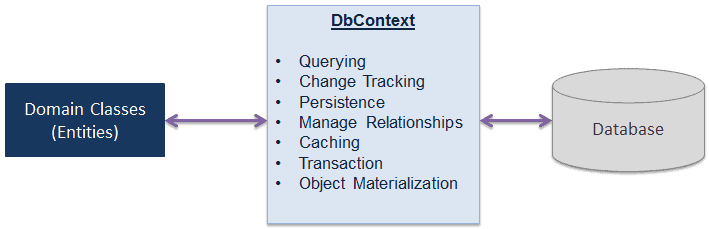
**Database First:** Database and Tables are there and based on them Model Classes and Database Context class are generated.

**DbContext is an important class in Entity Framework API. It is a bridge between your domain or entity classes and the database.**

**Entity Framework dbcontext**

**Reference URL:** <https://www.entityframeworktutorial.net/entityframework6/dbcontext.aspx#:~:text=DbContext%20is%20an%20important%20class,for%20interacting%20with%20the%20database.&text=Change%20Tracking%3A%20Keeps%20track%20of,after%20querying%20from%20the%20database>.

**DbContext** is the primary **class that is responsible for interacting with the database.** It is responsible for the following activities:



**Querying:** Converts LINQ-to-Entities queries to SQL query and sends them to the database.

**Change Tracking:** Keeps track of changes that occurred on the entities after querying from the database.

**Persisting Data:** Performs the Insert, Update and Delete operations to the database, based on entity states.

**Caching:** Provides first level caching by default. It stores the entities which have been retrieved during the life time of a context class.

**Manage Relationship:** Manages relationships using CSDL, MSL and SSDL in Db-First, and using fluent API configurations in Code-First approach.

**Object Materialization:** Converts raw data from the database into entity objects.

**DbSet:**

**Reference URL:** <https://www.learnentityframeworkcore5.com/dbset#:~:text=In%20Entity%20Framework%20Core%2C%20the,on%20the%20entity%20set>.

In Entity Framework Core, the DbSet represents the set of entities. In a database, a group of similar entities is called an Entity Set.

The DbSet enables the user to perform various operations like add, remove, update, etc. on the entity set.

Each entity type shows some DbSet properties to participate in CRUD operations.

In the working model, the DbContext represents the DbSet property of all the entities, and it keeps the collection of entities in memory.

**Operations of DbSet**

The DbSet is responsible for performing all the basic CRUD (Create, Read, Update and Delete) operations on each of the Entity.

The DbSet operations are used to change any property of the entity in the EF Core. The most essential methods of the DbSet are:

Querying Data

Adding Data

Modifying Data

Deleting Data

**Loading Related Entities in Entity Framework Core**

**Reference URLs:**

<https://www.mssqltips.com/sqlservertip/6241/eager-loading-in-entity-framework-core/#:~:text=Eager%20Loading%20%2D%20Related%20entities%20are,the%20navigation%20property%20is%20accessed>

<https://www.c-sharpcorner.com/article/eager-loading-lazy-loading-and-explicit-loading-in-entity-framework/>

All three terms –

- Eager Loading, Lazy Loading and Explicit Loading -- refer to the process of loading the related entities. They define when to load the related entities or child entities.

**Eager Loading -** Related entities are loaded as part of the initial query.

**Lazy Loading -** Related entities are loaded when the navigation property is accessed.

**Explicit Loading -** Related entities are loaded explicitly, not as part of the initial query, but at a later point of time.

**Eager Loading**

Eager Loading helps you to load all your needed entities at once; i.e., all your child entities will be loaded at single database call. This can be achieved, using the Include method, which returns the related entities as a part of the query and a large amount of data is loaded at once.

For example, you have a User table and a UserDetails table (related entity to User table), then you will write the code given below. Here, we are loading the user with the Id equal to userId along with the user details.

**User usr = dbContext.Users.Include(a => a.UserDetails).FirstOrDefault(a => a.UserId == userId);**

**Lazy Loading**

It is the default behavior of an Entity Framework, where a child entity is loaded only when it is accessed for the first time. It simply delays the loading of the related data, until you ask for it.

For example, when we run the query given below, UserDetails table will not be loaded along with the User table.

**User usr = dbContext.Users.FirstOrDefault(a => a.UserId == userId);**

It will only be loaded when you explicitly call for it, as shown below.

**UserDeatils ud = usr.UserDetails;** // UserDetails are loaded here

**Explicit Loading**

There are options to disable Lazy Loading in an Entity Framework. After turning Lazy Loading off, you can still load the entities by explicitly calling the Load method for the related entities. There are two ways to use Load method Reference (to load single navigation property) and Collection (to load collections), as shown below.

**User usr = dbContext.Users.FirstOrDefault(a => a.UserId == userId);**

**dbContext.Entry(usr).Reference(usr => usr.UserDetails).Load();**

**When to use what**

Use Eager Loading when the relations are not too much. Thus, Eager Loading is a good practice to reduce further queries on the Server.

Use Eager Loading when you are sure that you will be using related entities with the main entity everywhere.

Use Lazy Loading when you are using one-to-many collections.

Use Lazy Loading when you are sure that you are not using related entities instantly.

When you have turned off Lazy Loading, use Explicit loading when you are not sure whether or not you will be using an entity beforehand.

# Raw SQL Queries

**Reference URL:** <https://docs.microsoft.com/en-us/ef/core/querying/raw-sql>

Entity Framework Core allows you to drop down to raw SQL queries when working with a relational database. Raw SQL queries are useful if the query you want can't be expressed using LINQ. Raw SQL queries are also used if using a LINQ query is resulting in an inefficient SQL query. Raw SQL queries can return regular entity types or [keyless entity types](https://docs.microsoft.com/en-us/ef/core/modeling/keyless-entity-types) that are part of your model.

**LINQ with EF Core:**

**Reference URLs:**

<https://docs.microsoft.com/en-us/ef/core/querying/>

<https://www.entityframeworktutorial.net/querying-entity-graph-in-entity-framework.aspx>

**Querying Data**

Entity Framework Core uses Language-Integrated Query (LINQ) to query data from the database. LINQ allows you to use C# (or your .NET language of choice) to write strongly typed queries. It uses your derived context and entity classes to reference database objects. EF Core passes a representation of the LINQ query to the database provider. Database providers in turn translate it to database-specific query language (for example, SQL for a relational database). Queries are always executed against the database even if the entities returned in the result already exist in the context.