* Controlling DB creation and schema changes
* Adding relationships
* Interact with EF Core data model
* Views , stored procedures , raw SQL
* Using EF Core in tests

**EF is data access technology for .NET development ,  
is an ORM(Object Relational Mapper)**

Microsoft Object Relational Mapper

EF – EF6(2008 - 2013) -> EF Core(2016) -> EF Core 2(2017) -> EF Core 3(2019)

Typical ORM : Class and Tables are similar structure

EF Core : between Domain Classes and DB has Mappin layer and Database Schema

Can to work with Azure Cosmos DB SQL API a no-SQL DB

**ADD EF Core to project**

NuGet Packages start with **entityframeworkcore**

Microsoft.EntityFrameworkCore.SqlServer

-> Microsoft.EntityFrameworkCore.Relational

-> Microsoft.EntityFrameworkCore

**DBContext**

* Changes tracking
* DB Interaction tasks

Contains a set of DbSets that define the the structure and relation of the DB tables , appropriate to the models structure.

Tables name will match the DbSet name

In EF Core you must specify both Data Provider and Connection String

Development env **–** hardcode the connection string in DB context class

Production env and for more profession way –define connection string in startup file to inject it to dbcontext in run time.

* EF Core Migration API
* Migration File
* EF Core Migration to create a DB or DB scripts
* Reverse engineer and existing DB into classes and DbContext

Migration file help EF.C need to know how to

1. build queries to work with DB schema   
   Example : (form p in people select p).ToList() 🡪 SELECT \* from people
2. Convert data return from DB to objects
3. Modified data to the DB
4. Keep DB updating base on current model state

EF.C Basic Migration Workflow:  
Define/Change Model 🡪 Create Migration File 🡪 Apply Migration to DB or Script

**Add First Migration**

Migration Tools must be in an **executable** project

For Migrations Commands : NuGet package Microsoft.EntityFrameworkCore.Tools  
For Migrations APIs : NuGet package Microsoft.EntityFrameworkCore.Design(dependency of Tools)

With Package Manager Console we ca work with Entity Framework Core commands:

Add-Migration : look at DbContext 🡪 determine data model 🡪 create migration file to create or migrate DB

Parameters : name of migration

Add two files:

1. <current date>\_<migration name>
2. ContexModelSnapShot – this file tracking model state and keep DB state relevant , in the next migration EF read the current model state and compare it to ContexModelSnapShot data 🡪 change schema

**Using migration to script or directly create DB**

script-migration : CLI option to create an equivalent SQL script of the migration file operation to run on SQL server

Development env : recommend use update-database

Production env : recommend use script-migration

**Reverse Engineer and Existing DB**

* **Create DbContext & classes from database**
* **Updating model - not supported**
* **Transition to migration – not pretty**

PowerShell command : scaffold-dbcontext

Parameters: Provider and connection string are required

**EF.C Mappings to DB**

Conventions – default assumptions (property name = column name)

Override with fluent mapping – applay in DbContext using Fluent API

Override with Data Annotation – applay in entity

**nTn – for Enteties E1 E2**

1. E1 has member collection<E2>
2. Join Entity E1JoinE2 with members: E1Id , E2Id , E1 , E2
3. E2 has member collection<E1JoinE2>

**1T1 – for Enteties E1 E2**

1. E1 has member E2
2. E2 has member E1Id

**Migration for model changes**

add-migration <migration name> 🡪 create new migration file

update-database 🡪 update database with the new migration file