Entity Framework Core

Eng. Wael Hosny Radwan

Course Content

- Introduction to Entity Framework Core
- Entity Framework Core workflow
- Installing Entity Framework Core
- Database First Approach
- DbContext class & Entities classes
- Querying Entities
  - LINQ to Entity
  - Native SQL

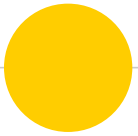
Course Content

- Querying Entities (Entity Graph)
  - DbSet Class
- Entity Lifecycle (Change Tracking)
  - DbEntityEntry Class
- Persistence in Entity Framework
  - Connected
  - Disconnected

Course Content

- Code First Approach
- Migrations in Entity Framework
- Conventions in Entity Framework
- Configuration Domain Classes
  - Data Annotation Attribute
  - Fluent API

Introduction to Entity Framework Core

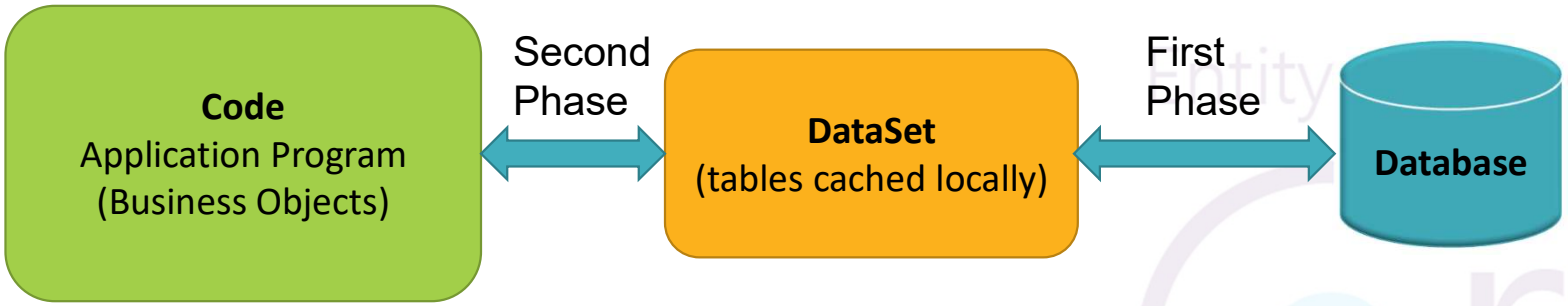


Check points

- Finalize (Destructor) and IDisposable

Before Entity Framework Core

- ADO.NET
  - Transforming data from Dataset to .NET objects (second phase) was the source of problems

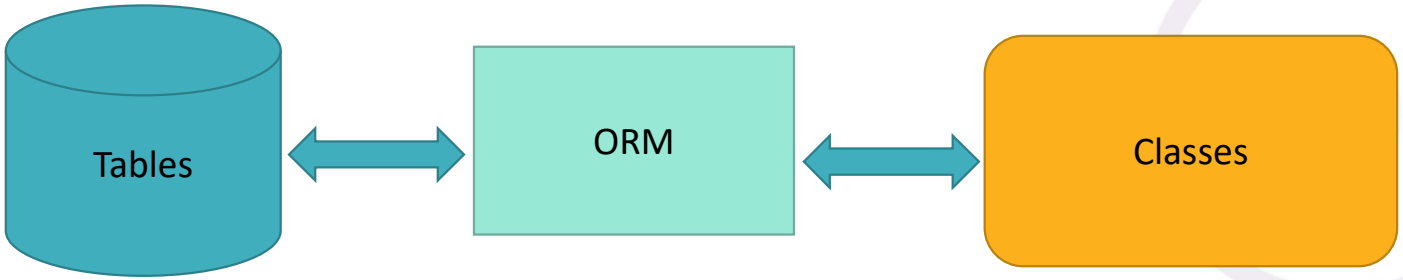


Before Entity Framework Core

- Entity Framework (.NET Framework 4.6 ...)

Before Entity Framework Core

- What is ORM (Object/Relational Mapping)
  - It is a tool for storing data from objects to tables in database
- ORM Components
  - Domain or business objects
  - Database objects (tables, relations ,constrains ,etc. …)
  - Information about mapping



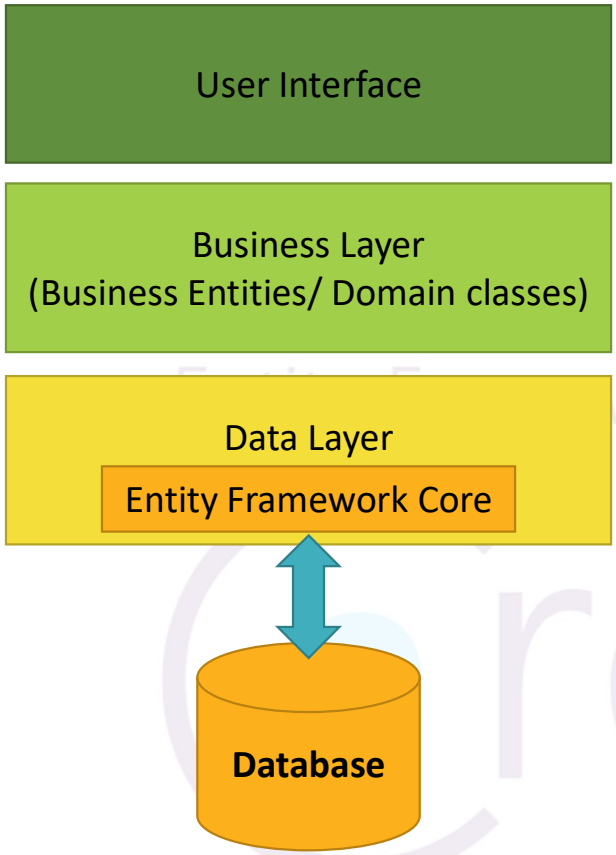
What is Entity Framework?

- The Microsoft ADO.NET Entity Framework is an Object/Relational Mapping (ORM) framework that enables developers to work with relational data as domain-specific objects, eliminating the need for most of the data access plumbing code that developers usually need to write. Using the Entity Framework, developers issue queries using LINQ, then retrieve and manipulate data as strongly typed objects. The Entity Framework's ORM implementation provides services like change tracking, identity resolution, lazy loading, and query translation so that developers can focus on their application-specific business logic rather than the data access fundamentals.

What is Entity Framework Core?

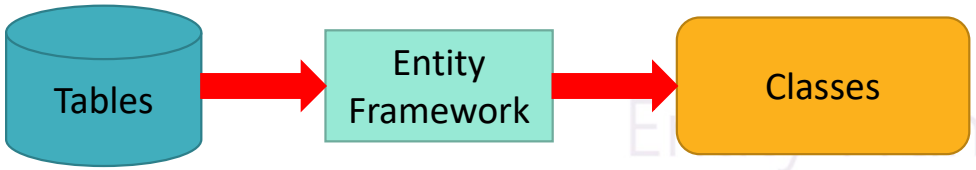
- Entity Framework Core is the new version of Entity Framework after EF 6.x. It is open-source, lightweight, extensible and a cross-platform version of Entity Framework data access technology.

Where Entity Framework in Application

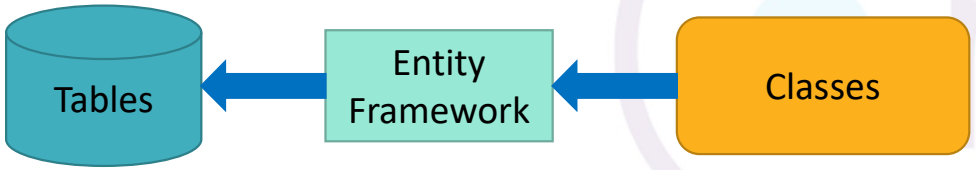


Entity Framework Approaches

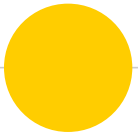
- Database first
  - Already have a database
  - Design the database first



- Code first
  - Focus on business objects and classes
  - Database generated from classes (entities)



Entity Framework Core Workflow



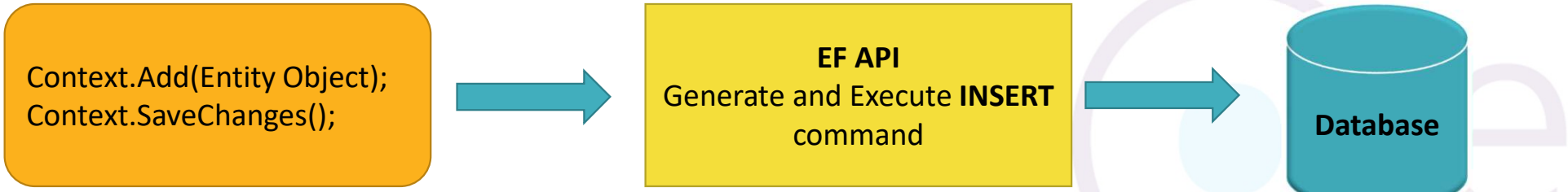
Entity Framework Basic workflow (CRUD operations)

- Define the Model
  - Domain classes
  - Context Class (DbContext)
  - Configurations

Entity Framework Basic workflow

(CRUD operations)

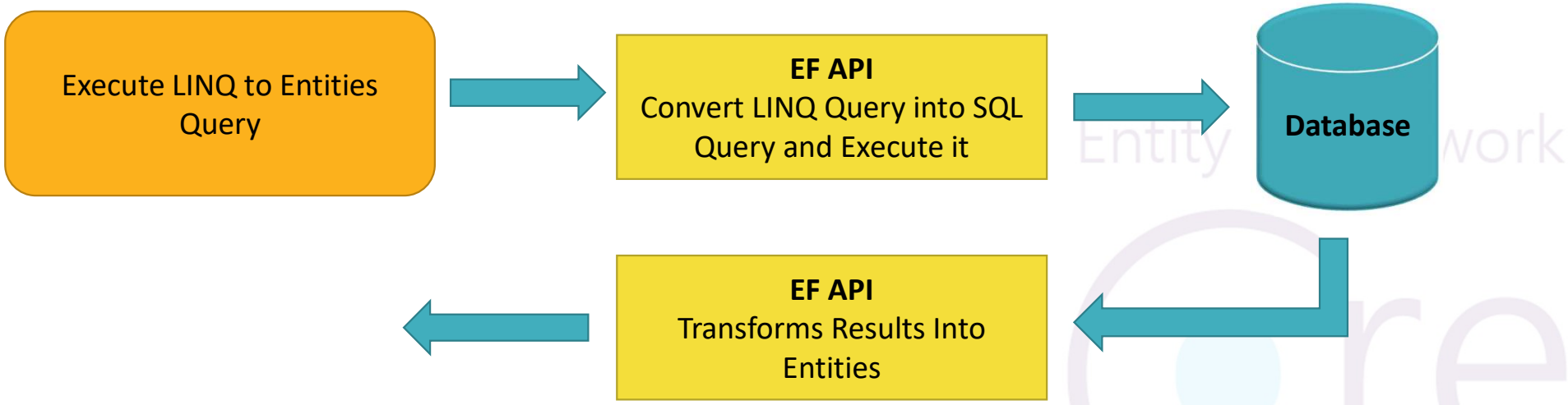
- Insert
  - Add Domain Object To Context
  - Call SaveChanges() method
  - EF API would generates appropriate INSERT Command and apply it to DB



Entity Framework Basic workflow

(CRUD operations)

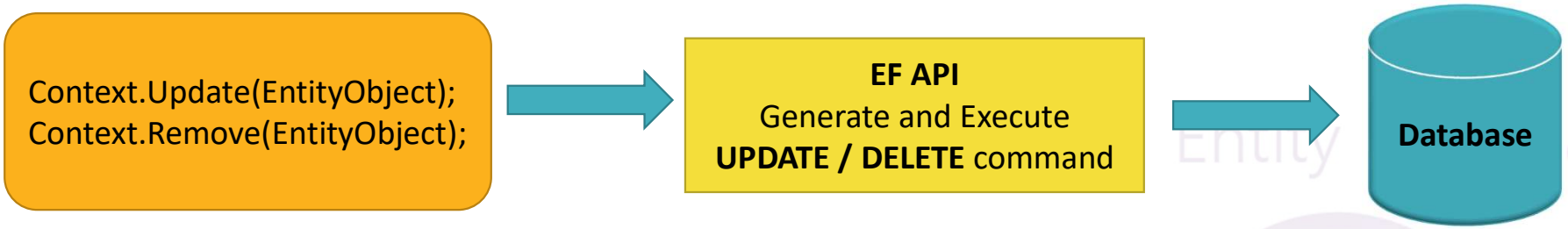
- Read



Entity Framework Basic workflow

(CRUD operations)

- Update or Delete



Entity Framework Core Responsibility

- Maps Classes to Database Schema
- Translate LINQ to Entities Queries to SQL Queries and Execute it
- Track Changes to Entities
- Save Changes to Database

Installing Entity Framework Core

Install Data Provider

- Data provider
  - plugin libraries used by EF to access many data base
- Add NuGet Package

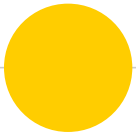
Database System	NuGet Package
SQL Server or Azure SQL	Microsoft.EntityFrameworkCore.SqlServer
Azure Cosmos DB	Microsoft.EntityFrameworkCore.Cosmos
SQLite	Microsoft.EntityFrameworkCore.Sqlite
EF Core in-memory database	Microsoft.EntityFrameworkCore.InMemory
PostgreSQL*	Npgsql.EntityFrameworkCore.PostgreSQL
MySQL/MariaDB*	Pomelo.EntityFrameworkCore.MySql
Oracle*	Oracle.EntityFrameworkCore

Installing EF Core Tools

- Add NuGet Package *Microsoft.EntityFrameworkCore.Tools*
  - For both (Code first, Database First)
  - Entity Framework Core Tools for the NuGet Package Manager Console in Visual Studio.
    - Add-Migration
    - Scaffold-DbContext
    - Update-Database
- Add NuGet Package *Microsoft.EntityFrameworkCore.Design*
  - (code first)
  - Used for creating database using migration

Database First Approach

(Reverse Engineering)



Using Visual Studio 2022

- Creating Entities and DbContext from Database
  - Using *scaffold-dbcontext* (Package Manager Console)

Connection string

```
Scaffold-DbContext
"Server=.\SQLExpress;Database=SchoolDB;Trusted_Connection=True;
Trust Server Certificate=true "
Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models
```

Data Provider

Output Directory

- Parameters
  - -Tables -DataAnnotations --Context -ContextDir

Using Visual Studio 2022

- SQLite

```
scaffold-dbcontext "data source=D:\DB\company.db"
microsoft.entityframeworkcore.sqlite -o Models
```

Using VS Code

- Terminal
  - Install EF globally `dotnet tool install --global dotnet-ef --version 9.*`

Connection string

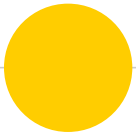
```
dotnet ef dbcontext scaffold
"Server=.\SQLExpress;Database=SchoolDB;Trusted_Connection=True;
Trust Server Certificate=true "
Microsoft.EntityFrameworkCore.SqlServer --output-dir Models
```

Data Provider

Output Directory

```
dotnet ef dbcontext scaffold "Data Source=School.db"
Microsoft.EntityFrameworkCore.Sqlite --context-dir Data --output-dir
Models
```

DbContext class & Entities classes



Generated Files

- File for each table in database contains type Class represent table for each table(Entity)
- File contains a class that inherits from DbContext (Context Class)

What is entities

- POCO classes (Plain-Old CLR Object)
- An entity is a class that maps database table
  - Contain properties represents each column in table

Scalar Properties

Navigation Properties

```
public partial class Employee
    {
        public int EmployeeId { get; set; }
        public string EmployeeName { get; set; }
        public int? EmployeeAge { get; set; }
        public double? EmployeeSalary { get; set; }
        public int? DepartmentId { get; set; }

        public virtual Department Department { get; set; }
    }
```

What is entities

Scalar Properties

Navigation Properties

```
public partial class Department
    {
        public int DepartmentId { get; set; }
        public string DepartmentName { get; set; }

        public virtual ICollection<Employee> Employees { get; set; } =
        new List<Employee>();
    }
```

What is entities

- Included as DbSet<TEntity> property in the Context class

```
public virtual DbSet<Department> Departments { get; set; }
public virtual DbSet<Employee> Employees { get; set; }
```

Entity Relationships (depends on tables)

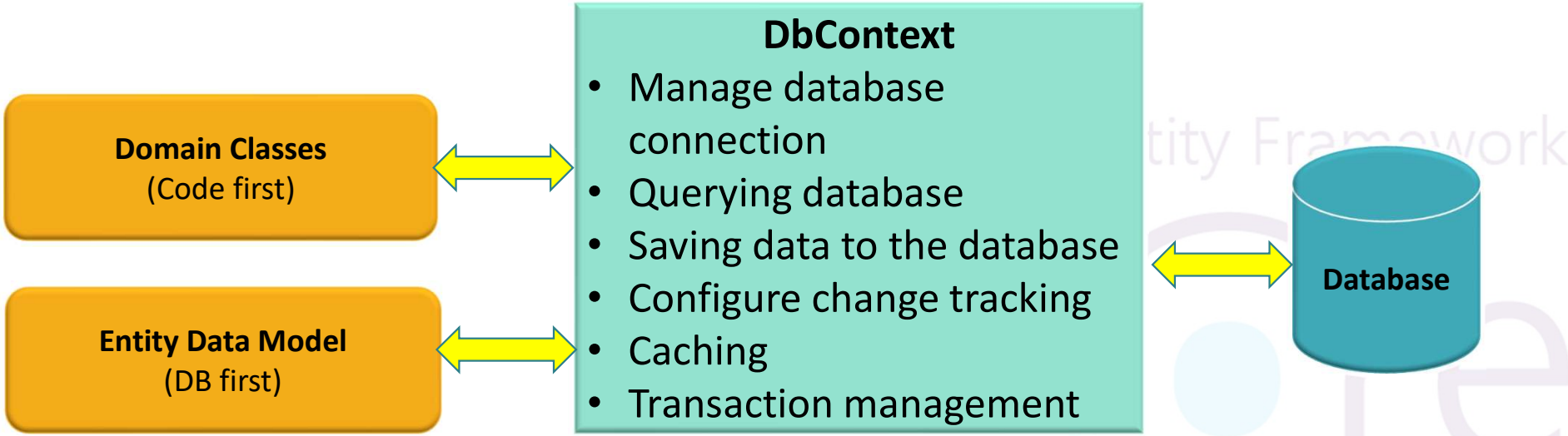
- One-to-One
  - each entity would contain reference to other entity
- One-to-Many
  - the one entity would contain a reference to collection of the the many entities
    - Ex: Department contains reference to collection of Employees
  - the many entity would contain reference to one entity
    - Ex: Employee contains reference to Department
- Many-to-Many
  - EF would create entity for the joining table

DbContext class

- The DbContext is a specialized base class that tracks our in-memory operations, allowing us to write and execute queries, track changes that we can persist back to our database
- An instance of DbContext represents a session with the database which can be used to query and save instances of your entities to a database

DbContext class

- DbContext in EF Core allows us to perform following tasks:



DbContext class

- A class Drived from DbContext

```
public partial class CompanyDBContext : DbContext
{
  public CompanyDBContext()
    {   }
  public CompanyDBContext(DbContextOptions<CompanyDBContext> options): base(options)
   {   }
    // Entities Collections
   public virtual DbSet<Department> Departments { get; set; }
   public virtual DbSet<Employee> Employees { get; set; }

   protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {      ... }
   protected override void OnModelCreating(ModelBuilder modelBuilder)
    {          ...}
}
```

DbContext class

- Constructor
  - Default

```
public CompanyDBContext():base()
{
}
```

    - Creating an object

```
public CompanyDBContext context= new CompanyDBContext();
```

DbContext class

- Constructor
  - String parameter (*connection string*)
    - Connection string (Server name, Database name, Username and password)

```
public class CompanyDBContext :DbContext
{
  readonly string _stringConn;
  public CompanyDBContext(string constr)
  {
     _stringConn=constr;
  }
```

DbContext class

- Constructor
  - DbContextOptions parameter

```
public CompanyDBContext(DbContextOptions<CompanyDBContext> options): base(options)
  {
  }
```

  - Creating an object

```
var contextOptions = new DbContextOptionsBuilder< CompanyDBContext >()
    .UseSqlServer(@"Server=(localdb)\mssqllocaldb;Database=Test;ConnectRetryCount=0")
    .Options;

using var context = new CompanyDBContext(contextOptions);
```

DbContext class

- `OnConfiguring(DbContextOptionsBuilder)` method
  - allows us to select and configure the *data source* to be used using *DbContextOptionsBuilder*
  - Called when creating an instance of DbContext with default constructor

```
public partial class CompanyDBContext : DbContext
{
  …
   protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder){

     optionsBuilder
.UseSqlServer("server=(localDb)\\MSSQLLocalDB;database=CompanyDB;Trusted_Connection=True;");
 }
}
```

DbContext class

- DbContextOptionsBuilder Methods

Database System	Example Configuration
SQL Server or Azure SQL	.UseSqlServer(connectionString)
Azure Cosmos DB	.UseCosmos(connectionString,databaseName)
SQLite	.UseSqlite(connectionString)
EF Core in-memory database	.UseInMemoryDatabase(databaseName)
PostgreSQL*	.UseNpgsql(connectionString)
MySQL/MariaDB*	.UseMySql(connectionString)
Oracle*	.UseOracle(connectionString)

DbContext class

- Using configuration JSON file
  - To make connection string in config file
  - add a .json file (e.g appconfig.json) at the root of your project and put the following content in it

```
{
  "ConnectionStrings": {
    "myDbConn":"server=(localdb)\\MSSQLLocalDB;Database=CompanyDB;Trusted_Connection=true"
  }
}
```

  - in the solution explorer, right click on the appconfig.json file and select Properties. Set the value of Copy to Output Directory to Copy Always.
  - install the *Microsoft.Extensions.Configuration.Json* package

DbContext class

- modify the `OnConfiguring` method
  - Install nugget package *Microsoft.Extensions.Configuration.json*

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    var config = new ConfigurationBuilder()
        .AddJsonFile("appconfig.json", optional: false).Build();

    if (!optionsBuilder.IsConfigured)
    {
        optionsBuilder.UseSqlServer(config.GetConnectionString("myDbConn"));
    }
}
```

DbContext class

- Using appsettings.json (***web application***)
- Modify `ConfigureServices` method on `startup` class

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<pubsContext>(options =>
    {
        options.UseSqlServer(Configuration.GetConnectionString(“myDbConn"));
    });
    services.AddControllersWithViews();
}
```

DbContext class

- OnModelCreating() Method
  - Allows us to tell Entity Framework Core more about the entities like:
    - Length of a property of an entitiy.
    - Whether a property is required by default.
    - Relationships between the entities. One-to-Many, One-to-One,
  - allows us to configure the model using ModelBuilder Fluent API.

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
    }
```

DbContext class

- Register `DbContext` as a services in application (ASP.NET)

DbContext class

- DbContext Methods

Method	Usage
Add	Adds a new entity to DbContext with Added state and starts tracking it. This new entity data will be inserted into the database when SaveChanges() is called.
AddAsync	Asynchronous method for adding a new entity to DbContext with Added state and starts tracking it. This new entity data will be inserted into the database when SaveChangesAsync() is called.
AddRange	Adds a collection of new entities to DbContext with Added state and starts tracking it. This new entity data will be inserted into the database when SaveChanges() is called.
AddRangeAsync	Asynchronous method for adding a collection of new entities which will be saved on SaveChangesAsync().

DbContext class

- DbContext Methods

Method	Usage
Attach	Attaches a new or existing entity to DbContext with Unchanged state and starts tracking it.
AttachRange	Attaches a collection of new or existing entities to DbContext with Unchanged state and starts tracking it.
Entry	Gets an EntityEntry for the given entity. The entry provides access to change tracking information and operations for the entity.
Find	Finds an entity with the given primary key values. Ex context.Find<Department>(1);
FindAsync	Asynchronous method for finding an entity with the given primary key values.
Remove	Sets Deleted state to the specified entity which will delete the data when SaveChanges() is called.

DbContext class

- DbContext Methods

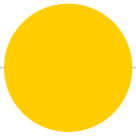
Method	Usage
RemoveRange	Sets Deleted state to a collection of entities which will delete the data in a single DB round trip when SaveChanges() is called.
SaveChanges	Execute INSERT, UPDATE or DELETE command to the database for the entities with Added, Modified or Deleted state.
SaveChangesAsync	Asynchronous method of SaveChanges()
Set	Creates and return DbSet<TEntity> that can be used to query and save instances of TEntity. Ex: var users = dbcontext.Set<User>();
Update	Attaches disconnected entity with Modified or Added (**depends on the value of the primary key**)state and start tracking it. The data will be saved when SaveChagnes() is called.
UpdateRange	Attaches a collection of disconnected entities with Modified state and start tracking it. The data will be saved when SaveChagnes() is called.

DbContext class

- DbContext Properties

Method	Usage
ChangeTracker	Provides access to information and operations for entity instances this context is tracking.
Database	Provides access to database related information and operations for this context.
Model	Returns the metadata about the shape of entities, the relationships between them, and how they map to the database.

Querying Entities
(Read)



Querying Entities

- LINQ to Entities
- Native SQL (Raw SQL Queries)
- Stored Procedure

Querying Entities

- DbSet implements ***IQuerayable*** Interface
  - So LINQ could be used to Query against DbSet
- DbSet Method
  - Find(ID)
- Native SQL

LINQ to Entities

- LINQ Method Syntax

```
Using(var ctx  = new SchoolDBEntities())// instantiate contex
{
  var L2EQuery = ctx.Students.Where(s => s.StudentName == "Bill"); // get the query result
  var student = L2EQuery.FirstOrDefault<Student>(); //get the first student in collection
}
```

- LINQ Query syntax

```
Using(var ctx  = new SchoolDBEntities())// instantiate contex
{
  var L2Query = from st in ctx. Students
                where st. StudentsName == "Bill"
                select st;
  var student = L2EQuery.FirstOrDefault<Student>(); //get the first student in collection
}
```

Native SQL

- Raw SQL Queries

```
Using(var ctx  = new SchoolDBEntities())// instantiate context
{
var studentName = ctx.Students.FromSqlRaw("Select studentid, studentname, standardId
from Student where studentname = 'Bill'").FirstOrDefault<Student>();
}
```

```
Using(var ctx  = new SchoolDBEntities())// instantiate context
{
var studentName = ctx.Students.FromSqlRaw("Select studentid, studentname, standardId
from Student where studentname ={0}",name).FirstOrDefault<Student>();
}
```

Stored Procedure

- Calling stored Procedure

```
ctx.FromSqlRaw("exec Select_Sp @ID={0}", 1).FirstOrDefault();
```

```
var IDParam=new SqlParameter("@ID", 1);
ctx.FromSqlRaw("exec Select_Sp @ID",IDParam).FirstOrDefault();
```

LINQ to Entities Queries

- GroupBy
  - LINQ Method

```
using(var ctx=new CompanyDBContext())
            {
              var query=  ctx.Employees.Include(e => e.Department)
                    .AsEnumerable()
                    .GroupBy(em => em.Department.DepartmentName);
                foreach(var grp in query)
                {
                    Console.WriteLine("===="+grp.Key+"=====");
                    foreach(var emp in grp)
                    {
                        Console.WriteLine(emp.EmployeeName);
                    }
                }
            }
```

LINQ to Entities Queries

- GroupBy
  - LINQ Query

```
using (var ctx = new CompanyDBContext())
            {
                var query = from em in ctx.Employees.Include(e => e.Department)
                            .AsEnumerable()
                            group em by em.Department.DepartmentName;
                foreach (var grp in query)
                {
                    Console.WriteLine("====" + grp.Key + "=====");
                    foreach (var emp in grp)
                    {
                        Console.WriteLine(emp.EmployeeName);
                    }
                }
            }
```

LINQ to Entities Queries

- OrderBy
  - LINQ Method

```
Using(var ctx = new CompanyDBContext())// instantiate contex
{
    var students = ctx.Employees.OrderBy(e => e.EmployeeName).ToList();
    var students = ctx.Employees. OrderByDescending(e => e.EmployeeName).ToList();
}
```

  - LINQ Query

```
Using(var ctx  = new SchoolDBEntities())// instantiate contex
{
var students = from s in ctx.Students
                          orderby s.StudentName ascending
                          select s;
}
```

LINQ to Entities Queries

- Anonymous Object:
  - LINQ Method

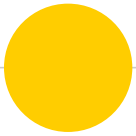
```
using(var ctx = new CompanyDBContext())// instantiate contex
 {
    var anonymousObjResult = ctx.Employees
                             .Where(em => em.DepartmentId == 1)
                             .Select(em => new { em.DepartmentId, em.EmployeeName });
    }
```

  - LINQ Query

```
Using(var ctx  = new SchoolDBEntities())// instantiate contex
{var anonymousObjResult = from em in ctx.Employees
                                     where em.DepartmentId == 1
                                     select new
                                     {  em.DepartmentId,em.EmployeeName };
}
```

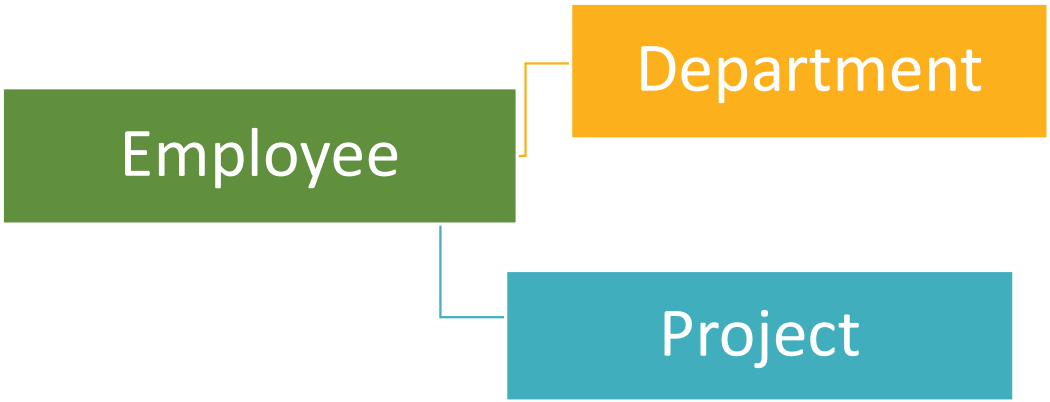
Querying Entities

(Read - Entity Graph)



Querying Entities (Related Data)

- Entity Graph
  - Represents the relations of the entity with other entities



Querying Entities (Related Data)

- Read Records
  - Eager loading
  - Explicit Loading
  - Lazy Loading

Querying Entities (Related Data)

- No related data loaded (*default behavior*)
  - Navigation property =*null (unless it was loaded before)*

```
using(var ctx = new CompanyDBContext())// instantiate context
 {
    var Emp = ctx.Employees.Where(em => em.DepartmentId == 1).FirstOrDefault();
      // Emp.Department = null
    var Dept = ctx.Departments. FirstOrDefault();
    // Dept.Employees = null

 }
```

Querying Entities (Related Data)

- Eager loading
  - Include()

```
using(var ctx = new CompanyDBContext())// instantiate context
 {
    var Emp = ctx.Employees
        .Include(e=>e.Department)
        .Where(em => em.DepartmentId == 1).FirstOrDefault();

    var Dept = ctx.Departments
                .Include(D=>D.Employees)
                .FirstOrDefault();
    }
```

  - Eager loading a collection navigation in a single query may cause performance issues

Querying Entities (Related Data)

- Eager Loading
  - Multiple Include()

```
using(var ctx = new CompanyDBContext())// instantiate context
 {
    var Dept = ctx.Departments
                .Include(D=>D.Employees)
                .Include(d=>d.Project)
                .FirstOrDefault();
 }
```

Querying Entities (Related Data)

- Eager Loading
  - Including multiple levels

```
using(var ctx = new CompanyDBContext())// instantiate context
 {

    var Dept = ctx.Departments
                .Include(D=>D.Employees)
                .ThenInclude(Emp=>emp.Adresses)
                .FirstOrDefault();

    }
```

Querying Entities (Related Data)

- Cartesian explosion Issue
  - `AsSplitQuery()`
    - Generate SQL Statement for each include instead using JOIN statement

```
using(var ctx = new CompanyDBContext())// instantiate context
 {
    var Emp = ctx.Employees
        .Include(e=>e.Deparment)
        .AsSplitQuery()
        .Where(em => em.DepartmentId == 1).FirstOrDefault();

    var Dept = ctx.Departments
               .Include(D=>D.Employees)
               .FirstOrDefault();
    }
```

Querying Entities (Related Data)

- Enabling split queries globally

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
   optionsBuilder.UseSqlServer("server=(localDb)\\MSSQLLocalDB;database=CompanyDB;
  Trusted_Connection=True;“,
            o => o.UseQuerySplittingBehavior(QuerySplittingBehavior.SplitQuery) );
 }
```

- One Query

```
using(var ctx = new CompanyDBContext())
 {     var Emp = ctx.Employees
         .Include(e=>e.Deparment)
         .AsSingleQuery()
         .Where(em => em.DepartmentId == 1).FirstOrDefault();
}
```

Querying Entities (Related Data)

- Explicit Loading
  - means that the related data is explicitly loaded from the database at a later time (through navigation property)
  - via `DbContext.Entry(...)` API.

```
using(var ctx = new CompanyDBContext())
 {
    var dept =
    ctx.Departments.FirstOrDefault();
        …
    ctx.Entry(dept)
    .Collection(d=>d.Employees)
    .Load();

    }
```

```
using(var ctx = new CompanyDBContext())
 {
    var Emp =
    ctx.Employees.FirstOrDefault();
    …
    ctx.Entry(Emp)
    .Reference(e=>e.Department)
    .Load();

    }
```

Querying Entities (Related Data)

- Explicit Loading
  - Querying related Data (ex: count , filtering)

```
using(var ctx = new CompanyDBContext())
 {
    var Dept =
    ctx.Departments.FirstOrDefault();

    ctx.Entry(dept)
    .Collection(d=>d.Employees)
    .Query()
    .Count();

    }
```

```
using(var ctx = new CompanyDBContext())
 {
    var Dept =
    ctx.Departments.FirstOrDefault();

    ctx.Entry(dept)
    .Collection(d=>d.Employees)
    .Query()
    .Where(e=>e.salary>2000);

    }
```

Querying Entities (Related Data)

- Lazy Loading
  - means that the related data is Automatically loaded from the database when the navigation property is accessed.

```
using(var ctx = new CompanyDBContext())// instantiate context
 {

    var Dept = ctx.Departments.FirstOrDefault();

    var Emp=Dept.Employees.ElementAt(0);

    }
```

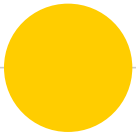
Querying Entities (Related Data)

- Lazy Loading With Proxies
  - Install NuGet package *Microsoft.EntityFrameworkCore.Proxies*
  - Make all navigation properties virtual
  - in `OnConfiguring`() method Add `UseLazyLoadingProxies()`

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder )
{
    optionsBuilder
    .UseLazyLoadingProxies()
    .UseSqlite(config.GetConnectionString("MyCon"));
}
```

Working with Entities

(Add-Modify-Delete)



DbSet Class

- Represent the entity set

Method Name	Return Type	Description
Add	EntityEntry<TEntity>	Adds the given entity to the context. When the changes are being saved, the entities in the Added states are inserted into the database. After thechanges are saved, the object state changes to Unchanged Example: dbcontext.Students.Add(studentEntity)
AsNoTracking<Entity>	IEnumerable<TEntity>	Returns a new query where the entities returned will not be cached in the DbContext. Entities returned as AsNoTracking, will not be tracked by DBContext. This will be significant performance boost for read only entities. Detached Example: var studentList =dbcontext.Students. AsNoTracking<Student>().ToList<Student>();

DbSet Class

Method Name	Return Type	Description
Attach(Entity)	EntityEntry<TEntity>	Attaches the given entity to the context in the Unchanged state Example: dbcontext.Students.Attach(studentEntity); Note : primary key ould be setted otherwise added state would be setted
Find(int)	Entity type	Uses the primary key value to attempt to find an entity tracked by the context. If the entity is not in the context then a query will be executed and evaluated against the data in the data source, and null is returned if the entity is not found in the context or in the data source. Note that the Find also returns entities that have been added to the context but have not yet been saved to the database. Example: Student studEntity = dbcontext.Students.Find(1);

DbSet Class

Method Name	Return Type	Description

Include	IIncludableQueryable<TEntity,TProperty>	Returns the included non generic LINQ to Entities query against a DbContext. (Inherited from DbQuery) Eager loading
		Example:
		var studentList = dbcontext.Students.Include("StudentAddress").ToList<Student>();
		var studentList = dbcontext.Students.Include(s=>s.StudentAddress).ToList<Student>();

Remove	Removed entity	Marks the given entity as Deleted. When the changes are saved, the entity is deleted from the database. The entity must exist in the context in some other state before this method is called. Example: dbcontext.Students.Remove(studentEntity);

DbSet Class

Method Name	Return Type	Description
FromSqlRaw	IQueryable<TEntity>	Creates a raw SQL query that will return entities in this set. By default, the entities returned are tracked by the context; this can be changed by calling AsNoTracking on theDbSqlQuery<TEntity> returned from this method. Example: var studentEntity = dbcontext.Students.SqlQuery("select * from student where studentid =1").FirstOrDefault<Student>();

Assignment

- Design Database CompanyDB as in SQL file attached
- Connect to db (database first)
- Add employee , Department ,Project to Database using Data input from User
- Let user choose to Display Department or Project or Employee