

ORM Fundamentals

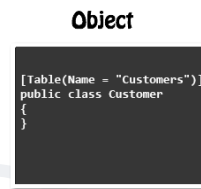
ADO.NET and ORM Frameworks



Microsoft®
ADO.NET



Relational DB



Object



ORM (Object-Relational Mapping) Frameworks
ADO.NET (Active Data Objects) Framework

SoftUni Team
Technical Trainers



SoftUni



Software University

<https://about.softuni.bg/>

sli.do

#csharp-db

- Traditional Data Access
- ORM Technologies
 - Basic Concepts
 - Advantages and Disadvantages
- ORM Features
 - Retrieving Entities from Database
 - Mapping Navigation Properties
 - Change Tracking
 - Generating SQL





ADO.NET

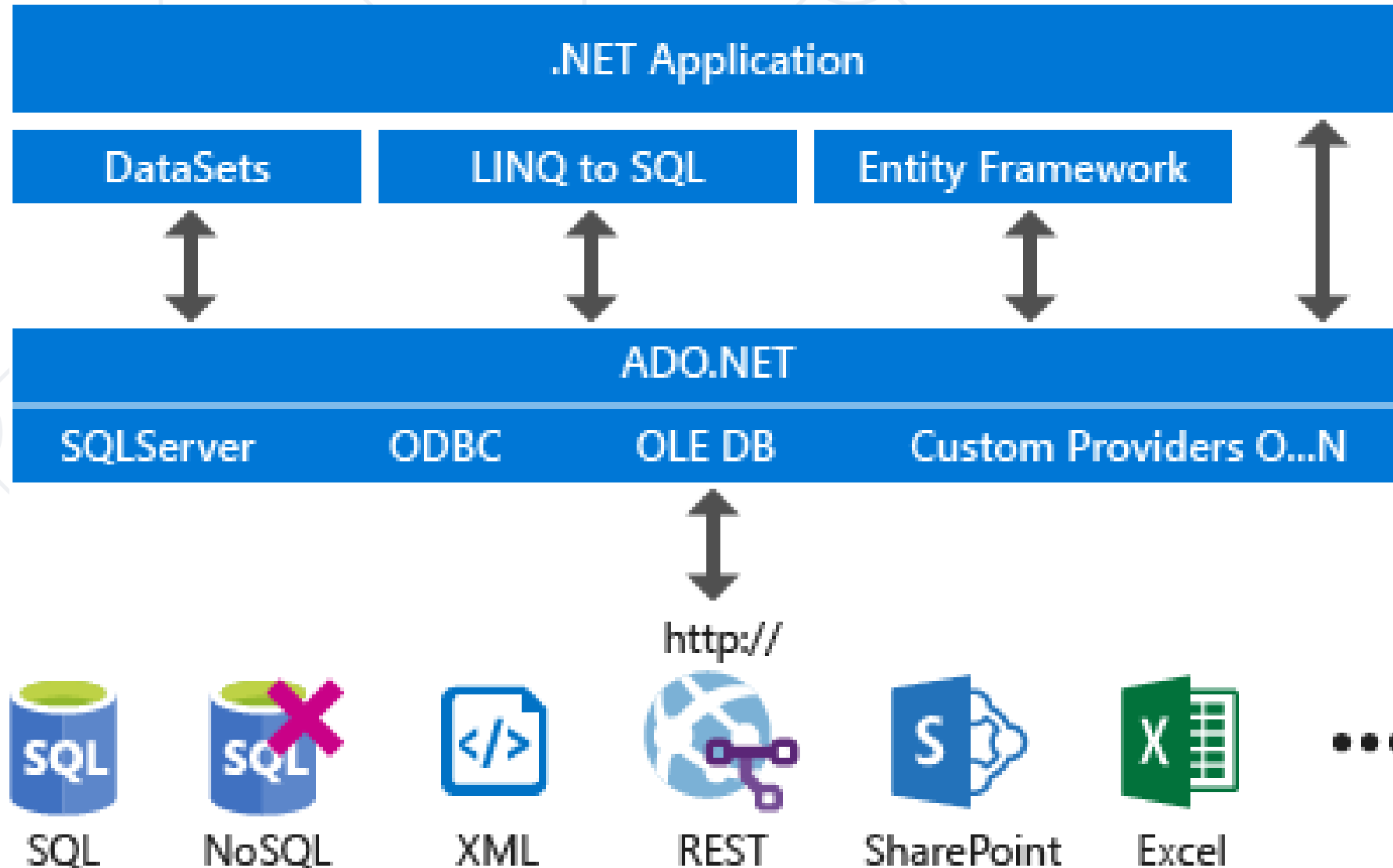
What is ADO.NET?

- **ADO.NET** is a standard **.NET class library** for accessing databases, processing data and XML
 - NuGet package for SQL Server: **Microsoft.Data.SqlClient**
 - <https://github.com/dotnet/SqlClient>
- Supports connected, disconnected and ORM data access models
 - Excellent integration with **LINQ**
 - Allows executing SQL in **RDBMS** systems
 - Allows accessing data in the **ORM** approach

- Data Providers are collections of classes that provide access to various databases
 - For different RDBMS systems different **Data Providers** are available
- Several common objects are defined
 - **Connection** – to connect to the database
 - **Command** – to run an SQL command
 - **DataReader** – to retrieve data

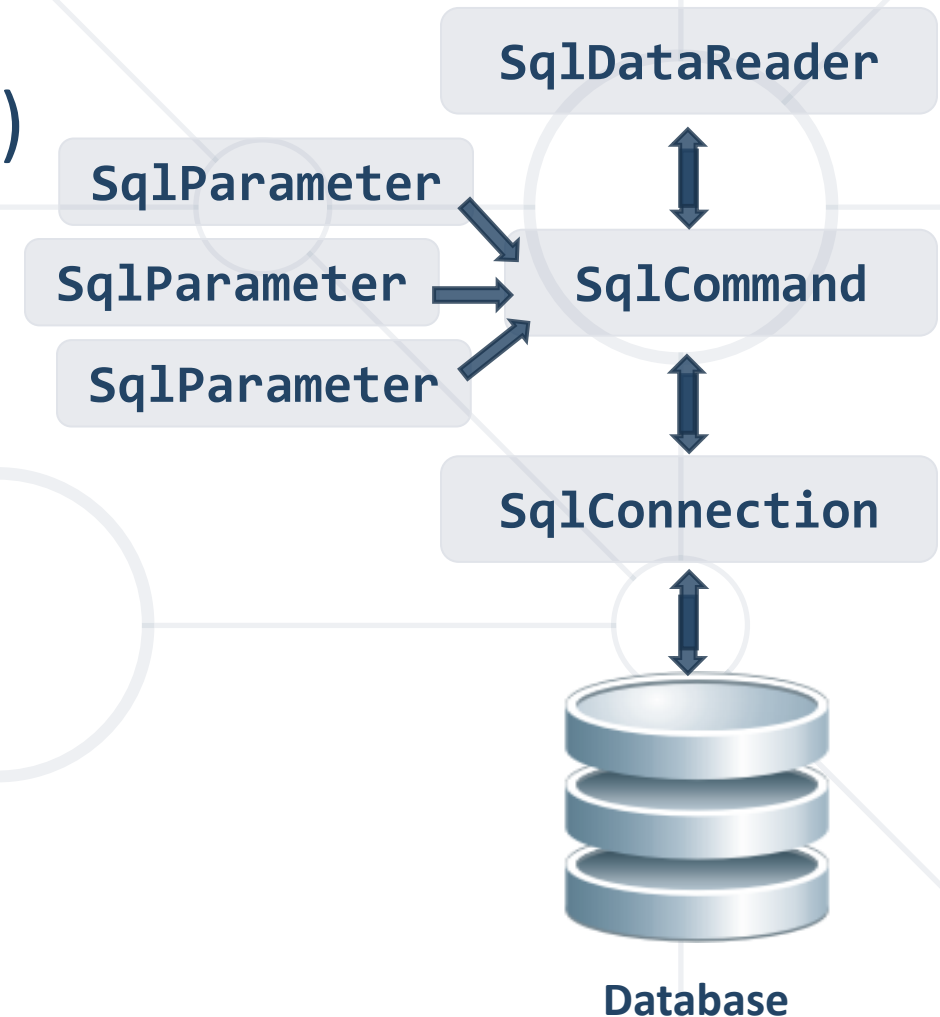
- Several standard ADO.NET Data Providers come as part of .NET Framework
 - **SqlClient** – accessing **SQL Server**
 - **OleDb** – accessing standard **OLE DB** data sources
 - **Odbc** – accessing standard **ODBC** data sources
 - **Oracle** – accessing **Oracle** databases
- Third party Data Providers are available for:
 - **MySQL, PostgreSQL, Interbase, DB2, SQLite**
 - Other RDBMS systems and data sources
 - SQL Azure, Salesforce CRM, Amazon SimpleDB, ...

.NET, EF, ADO.NET and Data Providers




SqlClient and ADO.NET Connected Model

- Retrieving data in connected model
 - Open a connection (**SqlConnection**)
 - Execute command (**SqlCommand**)
 - Process the result set of the query by using a reader (**SqlDataReader**)
 - Close the reader
 - Close the connection

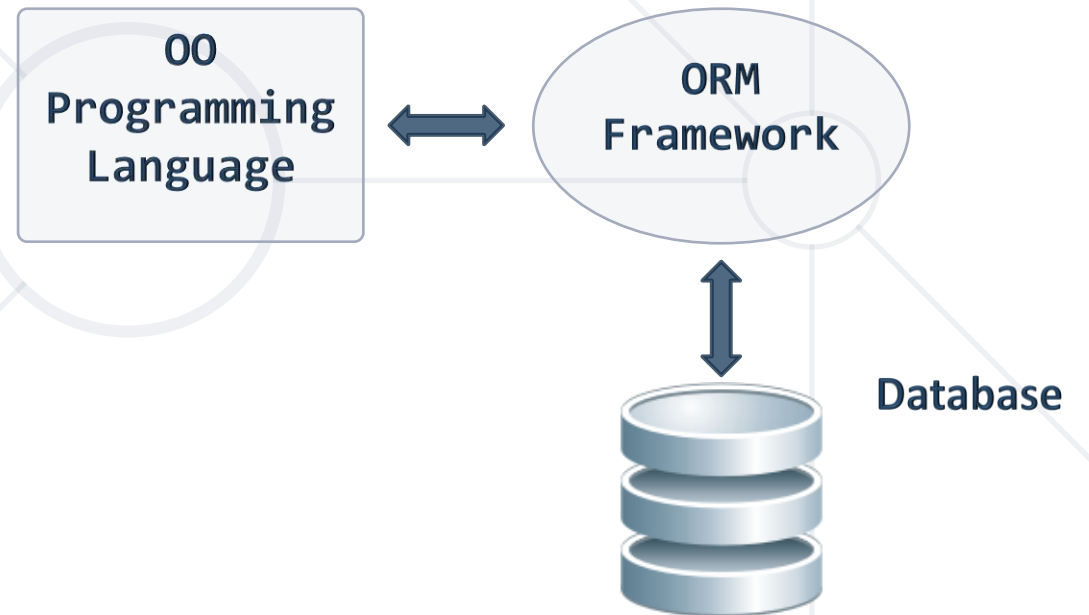


ORM (Object-Relational Mapping)

- **ORM data access model** (Entity Framework Core)
 - Maps **database tables** to **classes** and **objects**
 - Objects can be **automatically persisted** in the database
 - Can operate in both connected and disconnected modes

Employees	
	Id
	FirstName
	MiddleName
	LastName
	IsEmployed
	DepartmentId

```
public class Employee
{
    0 references
    public int Id { get; set; }
    0 references
    public string FirstName { get; set; } = null!;
    0 references
    public string LastName { get; set; } = null!;
    0 references
    public bool IsEmployed { get; set; }
    0 references
    public int DepartmentId { get; set; }
    0 references
    public Department Department { get; set; } = null!;
}
```



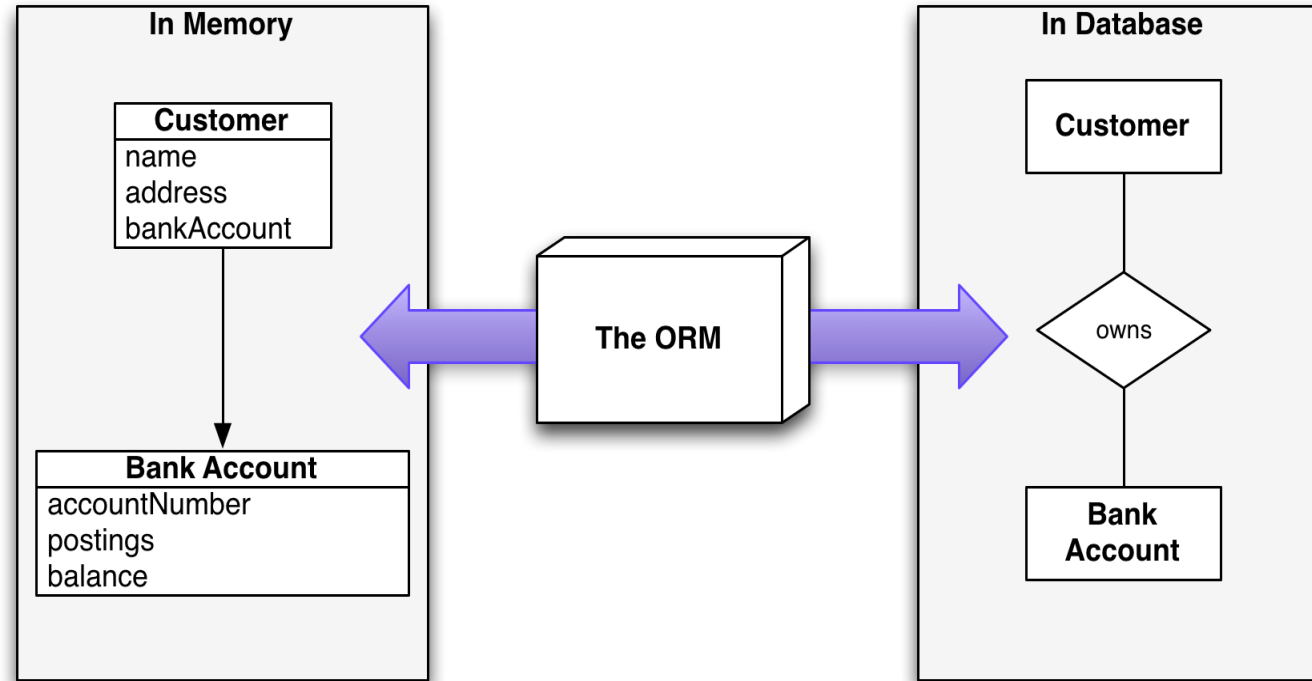
- **ORM benefits**

- Less code
- Use objects with **associations** instead of tables and SQL
- Integrated object query mechanism

- **ORM drawbacks**

- Less flexibility
 - SQL is automatically generated
- Performance issues (sometimes)

- **Entity Framework Core** is a generic **ORM** framework
 - Create entity data model mapping the database
 - Open an object context
 - Retrieve data with LINQ / modify the tables in the object context
 - Persist the object context changes into the DB
 - Connection is automatically managed




Introduction to ORM

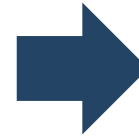
Object-Relational Mapping

What is ORM?

- **Object-Relational Mapping (ORM)** allows manipulating databases **using common classes and objects**
- **Database Tables → C#/Java/etc. classes**



Employees	
 Id	
FirstName	
MiddleName	
LastName	
IsEmployed	
DepartmentId	



```
public class Employee
{
    public int Id { get; set; }
    public string FirstName { get; set; }
    public string MiddleName { get; set; }
    public string LastName { get; set; }
    public bool IsEmployed { get; set; }
    public Department Department { get; set; }
}
```

- **ORM frameworks** typically **provide** the following functionality
 - **Automatically generate SQL** to perform data operations

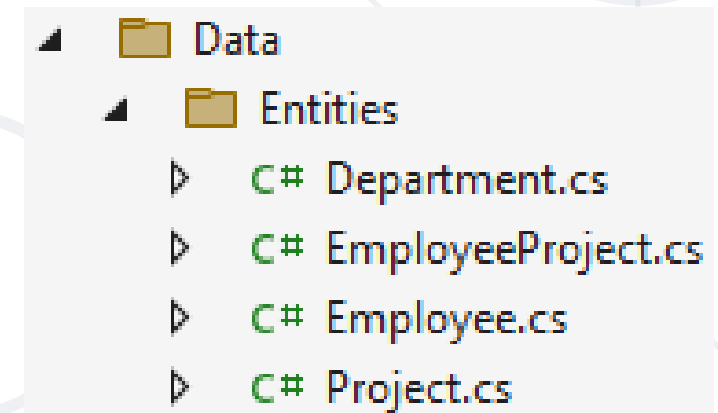
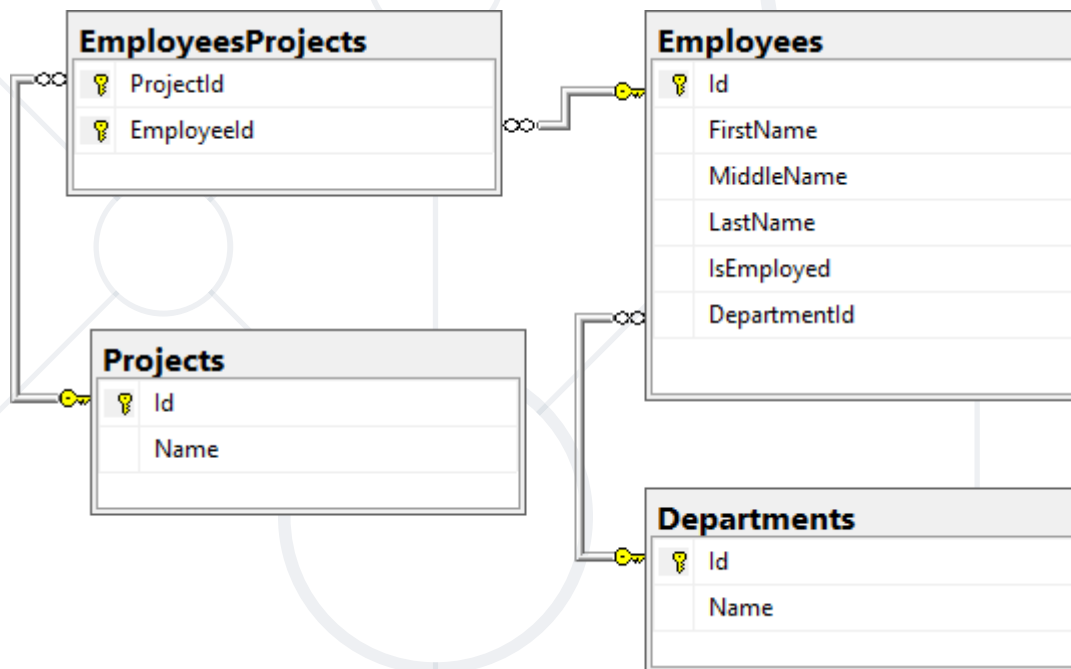
```
database.Employees.Add(new Employee  
{  
    FirstName = "George",  
    LastName = "Peterson",  
    IsEmployed = true    });
```



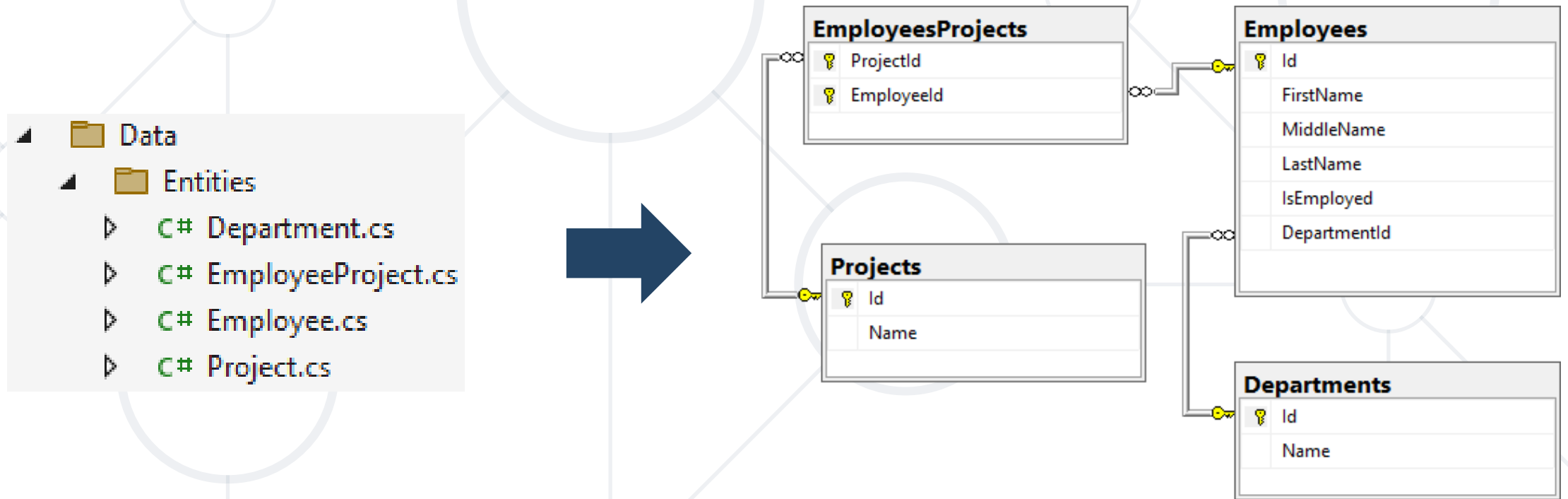
```
INSERT INTO Employees  
(FirstName, LastName, IsEmployed)  
VALUES  
( 'George ', 'Peterson', 1)
```

- **Create object model from database schema** (DB First model)
- **Create database schema from object model** (Code First model)
- **Query data by object-oriented API** (e.g., LINQ queries)

- **Database First model** - models the entity classes after the database



- **Code-first model** - begins with classes that describe the model and then the ORM generates a database

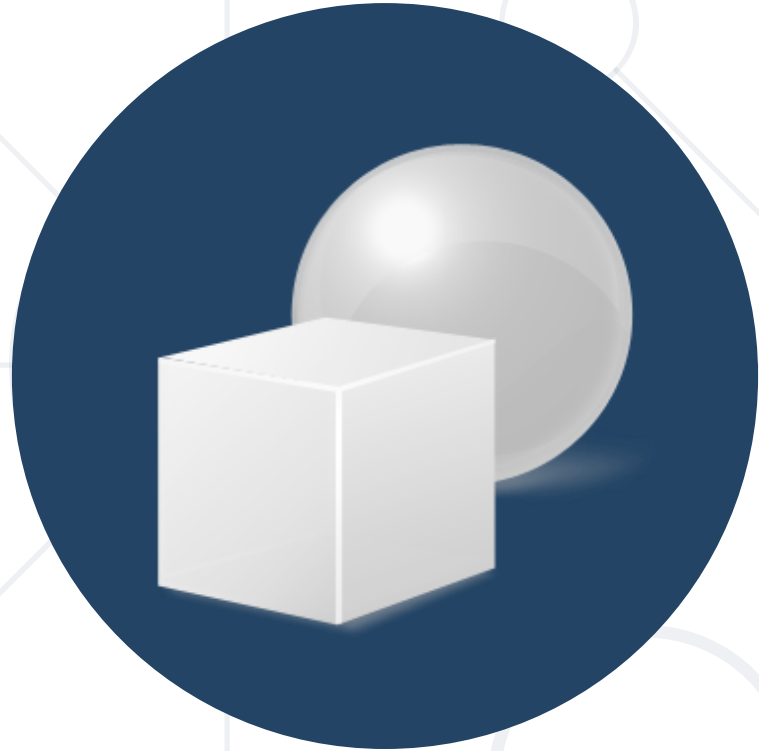


■ Advantages

- Developer productivity – **writing less code**
- Abstract from differences between object and relational world
- **Manageability of the CRUD operations** for complex relationships
- **Easier maintainability**

■ Disadvantages

- **Reduced performance** (due to overhead or autogenerated SQL)
- **Reduces flexibility** (some operations are hard to implement)





Entity Classes

Data Holders

Entity Classes

- **Entity classes** are regular **C# classes**
- Used for **storing** the **data** from the DB **in-memory**



Employees	
	Id
	FirstName
	MiddleName
	LastName
	IsEmployed
	DepartmentId



```
public class Employee
{
    public int Id { get; set; }
    public string FirstName { get; set; }
    public string MiddleName { get; set; }
    public string LastName { get; set; }
    public bool IsEmployed { get; set; }
    public Department Department { get; set; }
}
```

- Reference type properties
- Point to relevant object, connected by foreign key
- Set by the framework
- Example: Employee's Department

```
public class Employee
{
    public int Id { get; set; }
    ...
    public int DepartmentId { get; set; }
    public Department Department { get; set; }
}
```

- Navigation Properties can also be collections
- Usually of type **ICollection<T>**
- Hold all of the objects whose **foreign keys** are the same as the entity's **primary key**
- Set by the ORM framework

```
public class Department
{
    public int Id { get; set; }
    public ICollection<Employee> Employees { get; set; }
}
```



DbSet<T>

Specialized Collections

DbSet<T> Class

- Generic collection with additional features
- Each **DbSet<T>** corresponds to a single database table
- Inherits from **ICollection<T>**
 - **foreach**-able
 - Supports **LINQ** operations
- Usually several **DbSets** are a part of a **DbContext**



- Each **DbSet** tracks its own entities through a change tracker
- Has every other feature of an **ICollection<T>**
 - **Accessing** the elements (LINQ)
 - **Adding/Updating** elements
 - **Removing** an entity/a range of entities
 - **Checking** for element **existence**
 - Accessing the **count** of elements



DbContext

DbContext Class

- Holds several **DbSet<T>**
- Responsible for **populating** the **DbSets**
- Users create a **DbContext**, which **inherits** from **DbContext**
 - Using one **DbSet** per database table

```
public class SoftUniDbContext : DbContext
{
    public DbSet<Employee> Employees { get; set; }
    public DbSet<Department> Departments { get; set; }
    public DbSet<Project> Projects { get; set; }
}
```





Reading Data

Querying the DB Using ORM

- First create instance of the **DbContext**

```
var context = new SoftUniDbContext();
```

- In the constructor, you can pass a database connection string
- **DbContext properties**
 - All **entity classes** (tables) are listed as **properties**
 - e.g., **DbSet<Employee> Employees { get; }**

- Executing **LINQ-to-Entities** query over entity

```
var context = new SoftUniDbContext()

var employees = context.Employees
    .Where(e => e.JobTitle == "Design Engineer")
    .ToList();
```

- **Employees** property in the **DbContext**

```
public class SoftUniDbContext : DbContext
{
    public DbSet<Employee> Employees { get; }
    public DbSet<Project> Projects { get; }
    public DbSet<Department> Departments { get; }
}
```

- We can also use **extension methods** for constructing the query

```
var context = new SoftUniDbContext()
var employees = context.Employees
    .Where(c => c.JobTitle == "Design Engineering")
    .Select(c => c.FirstName)
    .ToList();
```

- Find element by **ID**

```
var context = new SoftUniEntities()
var project = context.Projects
    .FirstOrDefault(p => p.Id == 2);
Console.WriteLine(project.Name);
```

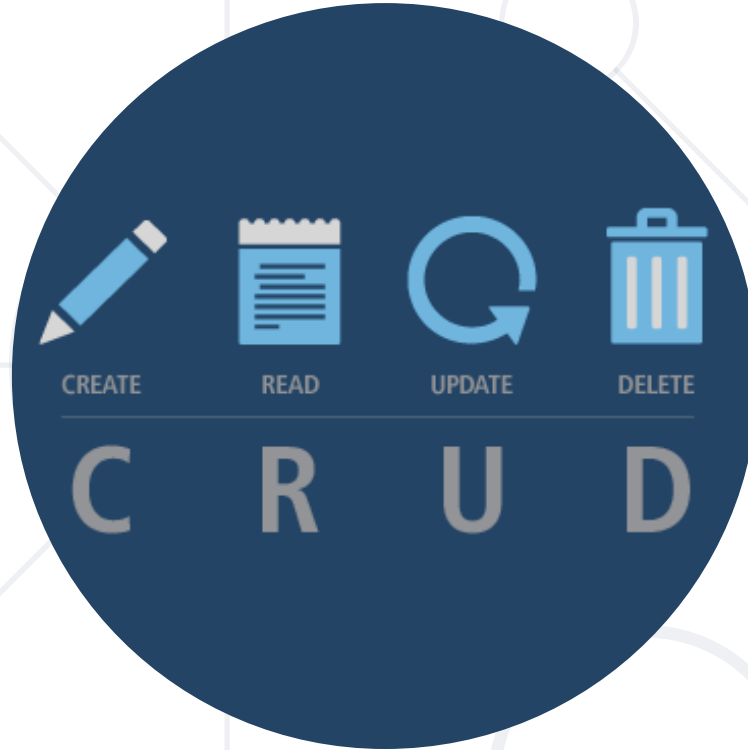


Change Tracking

Change Tracking

- Each **DbContext** instance tracks changes made to entities
 - These tracked entities in turn drive the changes to the database when **SaveChanges** is called
- Entity instances become tracked when they are
 - Returned from a query, executed against the database
 - Explicitly attached to the **DbContext** by **Add**, **Attach**, **Update** or similar methods
 - Detected as new entities connected to existing tracked entities





CRUD Operations

- To create a new table row use the method **Add(...)** of the **corresponding DbSet**

```
var project = new Project()  
{  
    Name = "Judge System"  
};
```

Create a new
Project object

Add the object to the DbSet

```
context.Projects.Add(project);  
context.SaveChanges();
```

Execute SQL statements

- **DbContext** allows modifying entity properties and persisting them in the database
 - Just load an entity, modify it and call **SaveChanges()**
- The **DbContext** automatically tracks all **changes** made on its entity **objects**

```
var employee =  
    context.Employees.FirstOrDefault();  
employee.FirstName = "Alex";  
context.SaveChanges();
```

SELECT the first
employee

Execute an SQL
UPDATE

Deleting Existing Data

- Delete is done by **Remove()** on the specified entity collection
- **SaveChanges()** method performs the delete action in the database

```
var employee =  
    context.Employees.First();  
context.Employees.Remove(employee);  
context.SaveChanges();
```

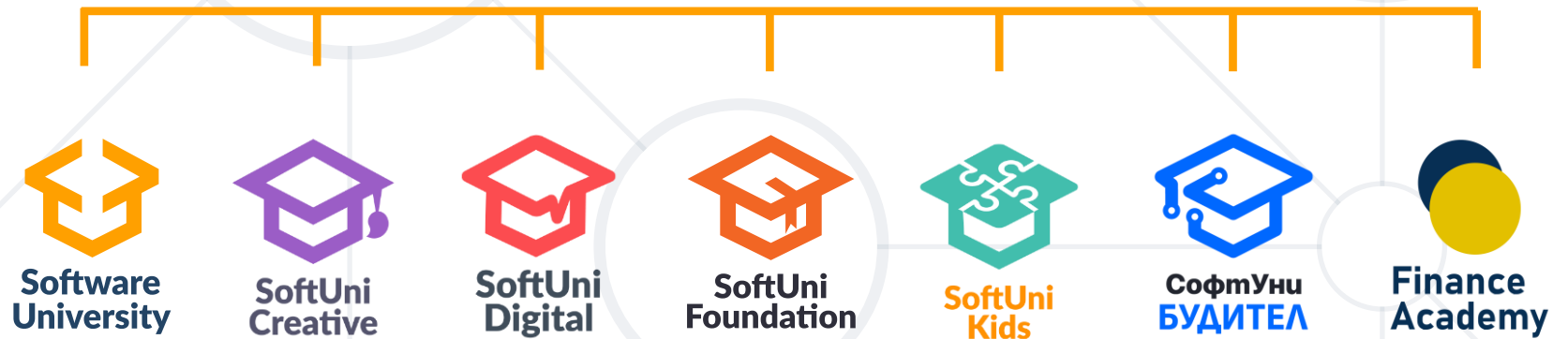
Mark the entity for deleting at the next save

Execute the SQL DELETE command

- **ADO.NET** - Primary Data Access
- **ORM frameworks** map database schema to objects in a programming language
- **LINQ** can be used to query the DB through the **DB Context**
 - Change Tracking
 - CRUD



Questions?



SoftUni Diamond Partners



- Software University – High-Quality Education, Profession and Job for Software Developers

- softuni.bg, about.softuni.bg

- Software University Foundation

- softuni.foundation

- Software University @ Facebook

- facebook.com/SoftwareUniversity



- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – <https://about.softuni.bg>
- © Software University – <https://softuni.bg>

