

ADO.NET



SoftUni Team
Technical Trainers



SoftUni



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

Table of Contents

- ADO.NET
- Accessing SQL Server from ADO.NET
- SQL Injection



Have a Question?



sli.do

#csharp-db

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

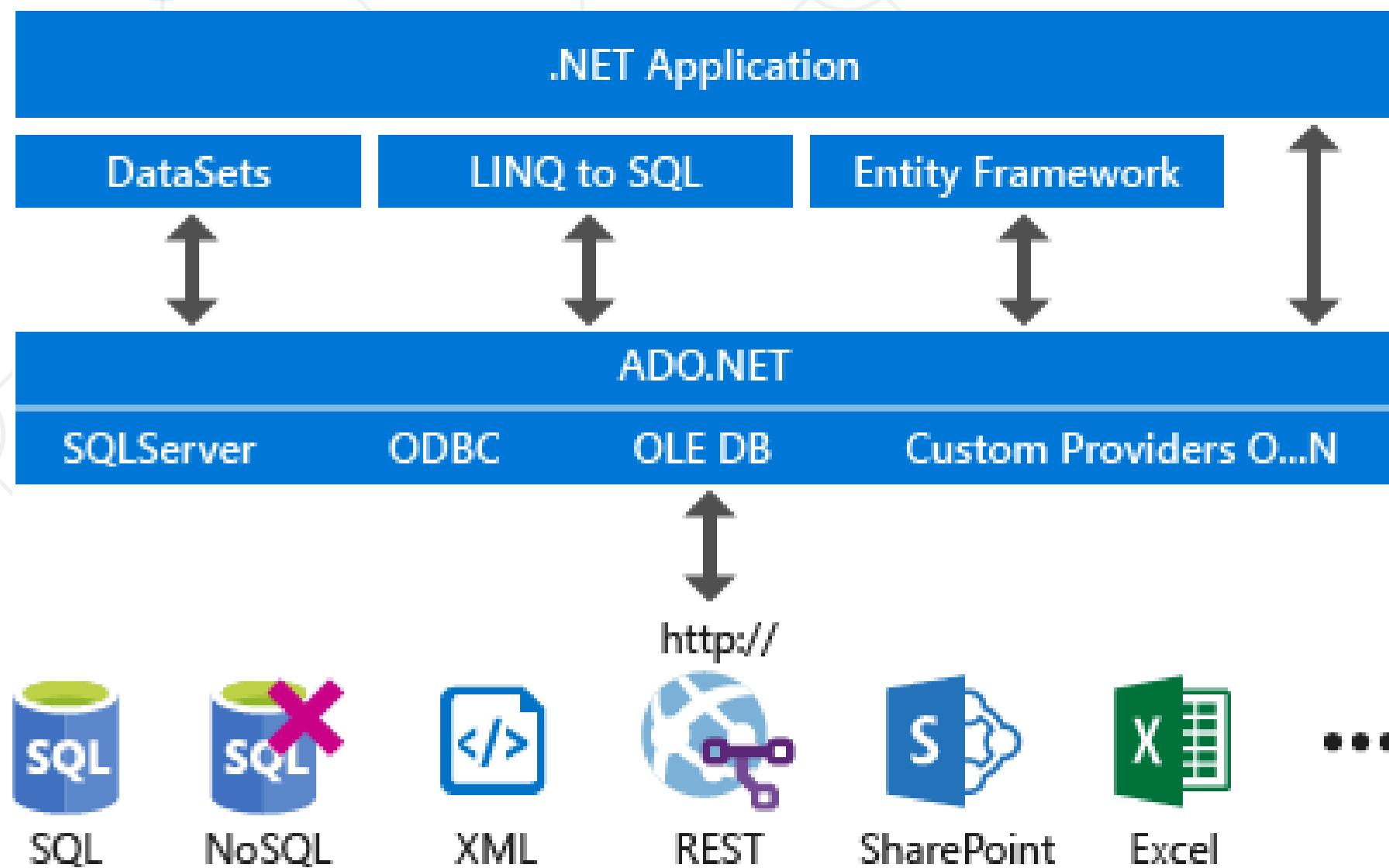
Data Providers in ADO.NET (1)

- 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

Data Providers in ADO.NET (2)

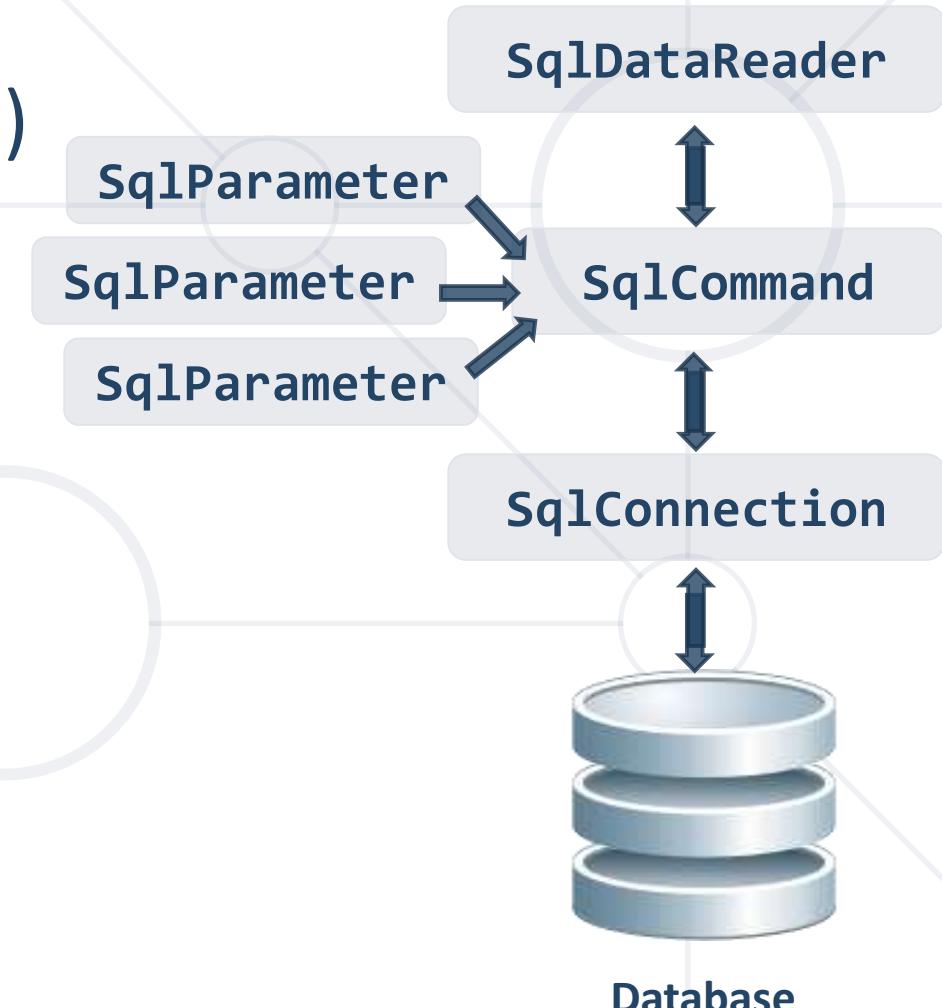
- Several standard ADO.NET Data Providers come as part of .NET Framework
 - **SqlClient** – accessing **SQL Server**
 - **OleDb** – accessing standard **OLE DB** data sources
 - **Odbc** – accessing standard **ODBC** data sources
 - **Oracle** – accessing **Oracle** databases
- Third party Data Providers 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



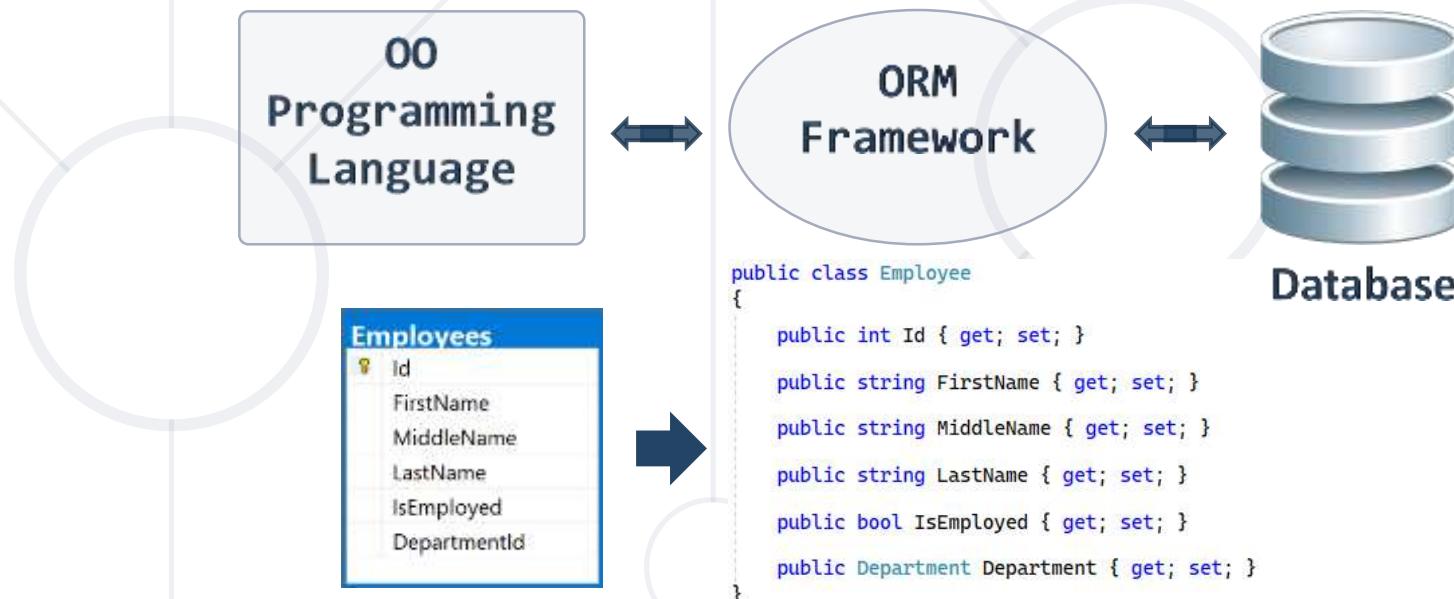
SqlClient and ADO.NET Connected Model

- 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



- **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



Accessing SQL Server from ADO.NET

SqlClient Data Provider

- **SqlConnection**
 - Establishes database connection to SQL Server
- **SqlCommand**
 - Executes SQL commands on the SQL Server through an established connection
 - Could accept parameters (**SqlParameter**)
- **SqlDataReader**
 - Retrieves data (record set) from SQL Server as a result of SQL query execution

The SqlConnection Class

- **SqlConnection** establishes a connection to SQL Server database
 - Requires a valid **connection string**
- Connection string example

```
Server=(local)\SQLEXPRESS;Initial  
Catalog=SoftUni;Integrated Security=true;
```

- Connecting to SQL Server

```
SqlConnection con = new SqlConnection(  
    @"Server=.;  
    Database=SoftUni;  
    Integrated Security=true");  
con.Open();
```

- **Database connection string**
 - Defines the parameters needed to establish the connection to the database
- **Settings for SQL Server connections**
 - **Data Source / Server** – server name / IP address + database instance name
 - **Database / Initial Catalog** – database name
 - **User ID / Password** – credentials
 - **Integrated Security** – false if credentials are provided

SqlConnection – Example

- Creating and opening connection to SQL Server (database **SoftUni**)

```
SqlConnection dbCon = new SqlConnection(  
    "Server=.\SQLEXPRESS; " +  
    "Database=SoftUni; " +  
    "Integrated Security=true");  
  
dbCon.Open();  
using (dbCon)  
{  
    // TODO: Use the connection to execute SQL commands here...  
}
```

Working with SqlConnection

- Explicitly opening and closing a connection
 - `Open()` and `Close()` methods
 - Works through the connection pool
- DB connections are `IDisposable` objects
 - Always use the `using` construct in C#!



The SqlCommand Class

- More important methods
 - **ExecuteScalar()**
 - Returns a single value - the value in the first column of the first row of the result set (as **System.Object**)
 - **ExecuteReader()**
 - Returns a **SqlDataReader**
 - It is a cursor over the returned records (result set)
 - **CommandBehavior** – assigns some options
 - **ExecuteNonQuery()**
 - Used for non-query SQL commands, e.g. **INSERT, UPDATE, DELETE, CREATE**
 - Returns the number of affected rows (**int**)

SqlCommand – Example

```
SqlConnection dbCon = new SqlConnection(  
    "Server=.; " +  
    "Database=SoftUni; " +  
    "Integrated Security=true");  
dbCon.Open();  
using(dbCon)  
{  
    SqlCommand command = new SqlCommand(  
        "SELECT COUNT(*) FROM Employees", dbCon);  
    int employeesCount = (int) command.ExecuteScalar();  
    Console.WriteLine("Employees count: {0}", employeesCount);  
}
```

The SqlDataReader Class

- **SqlDataReader** retrieves a sequence of records (cursor) returned as result of an SQL command
 - Data is available for reading-only (can't be changed)
 - Forward-only row processing (no move back)
- Important properties and methods
 - **Read()** – moves the cursor forward and returns **false**, if there is no next record
 - **Indexer[]** – retrieves the value in the current record by given column name or index
 - **Close()** – closes the cursor and releases resources

SqlDataReader – Example

```
SqlConnection dbCon = new SqlConnection(...);  
dbCon.Open();  
using(dbCon)  
{  
    SqlCommand command = new SqlCommand("SELECT * FROM Employees", dbCon);  
    SqlDataReader reader = command.ExecuteReader();  
    using (reader)  
    {  
        while (reader.Read())  
        {  
            string firstName = (string)reader["FirstName"];  
            string lastName = (string)reader["LastName"];  
            decimal salary = (decimal)reader["Salary"];  
            Console.WriteLine("{0} {1} - {2}", firstName, lastName, salary);  
        }  
    }  
}
```

Fetch more rows until finished



SQL Injection

What is SQL Injection? How to Prevent It?

What is SQL Injection? (1)

```
bool IsPasswordValid(string username, string password)
{
    string sql =
        $"SELECT COUNT(*) FROM Users " +
        $"WHERE UserName = '{username}' AND" +
        $"PasswordHash = '{CalcSHA1(password)}'";
    SqlCommand cmd = new SqlCommand(sql, dbConnection);

    int matchedUsersCount = (int)cmd.ExecuteScalar();
    return matchedUsersCount > 0;
}
```

What is SQL Injection? (2)

```
bool normalLogin =  
    IsPasswordValid("peter", "qwerty123"); // true  
  
bool sqlInjectedLogin =  
    IsPasswordValid(" ' or 1=1 --", "qwerty123"); // true  
  
bool evilHackerCreatesNewUser =  
    IsPasswordValid("' INSERT INTO Users VALUES('hacker','') --",  
    "qwerty123");
```

How Does SQL Injection Work?

- The following SQL commands are executed

- Usual password check (no SQL injection)

```
SELECT COUNT(*) FROM Users WHERE UserName = 'peter'  
AND PasswordHash = 'X0wXWxZePV5kiyeE86Ejvb+rIG/8='
```

- SQL-injected password check

```
SELECT COUNT(*) FROM Users WHERE UserName = ' ' or 1=1  
-- ' AND PasswordHash = 'X0wXWxZePV5kiyeE86Ejvb+rIG/8='
```

- SQL-injected INSERT command

```
SELECT COUNT(*) FROM Users WHERE UserName = ''  
INSERT INTO Users VALUES('hacker', '')  
-- ' AND PasswordHash = 'X0wXWxZePV5kiyeE86Ejvb+rIG/8='
```

Preventing SQL Injection

- Ways to prevent the SQL injection
 - SQL-escape all data coming from the user

```
string escapedUsername = username.Replace("'", "''");  
string sql =  
    "SELECT COUNT(*) FROM Users " +  
    "WHERE UserName = '" + escapedUsername + "' and " +  
    "PasswordHash = '" + CalcSHA1(password) + "'";
```
 - Not recommended: use as last resort only!
 - Preferred approach
 - Use **parameterized queries**
 - Separate the SQL command from its arguments

The SqlParameter Class

- What are **SqlParameters**?
 - SQL queries and stored procedures can have input and output parameters
 - Accessed through the **Parameters** property of the **SqlCommand** class
- Properties of **SqlParameter**
 - **ParameterName** – name of the parameter
 - **DbType** – SQL type (**NVarChar**, **Timestamp**, ...)
 - **Size** – size of the type (if applicable)
 - **Direction** – input / output

Parameterized Commands – Example

```
void InsertProject(string name, string description, DateTime startDate)
{
    SqlCommand cmd = new SqlCommand(
        "INSERT INTO Projects " +
        "(Name, Description, StartDate, EndDate) VALUES " +
        "(@name, @desc, @start, @end)", dbCon);

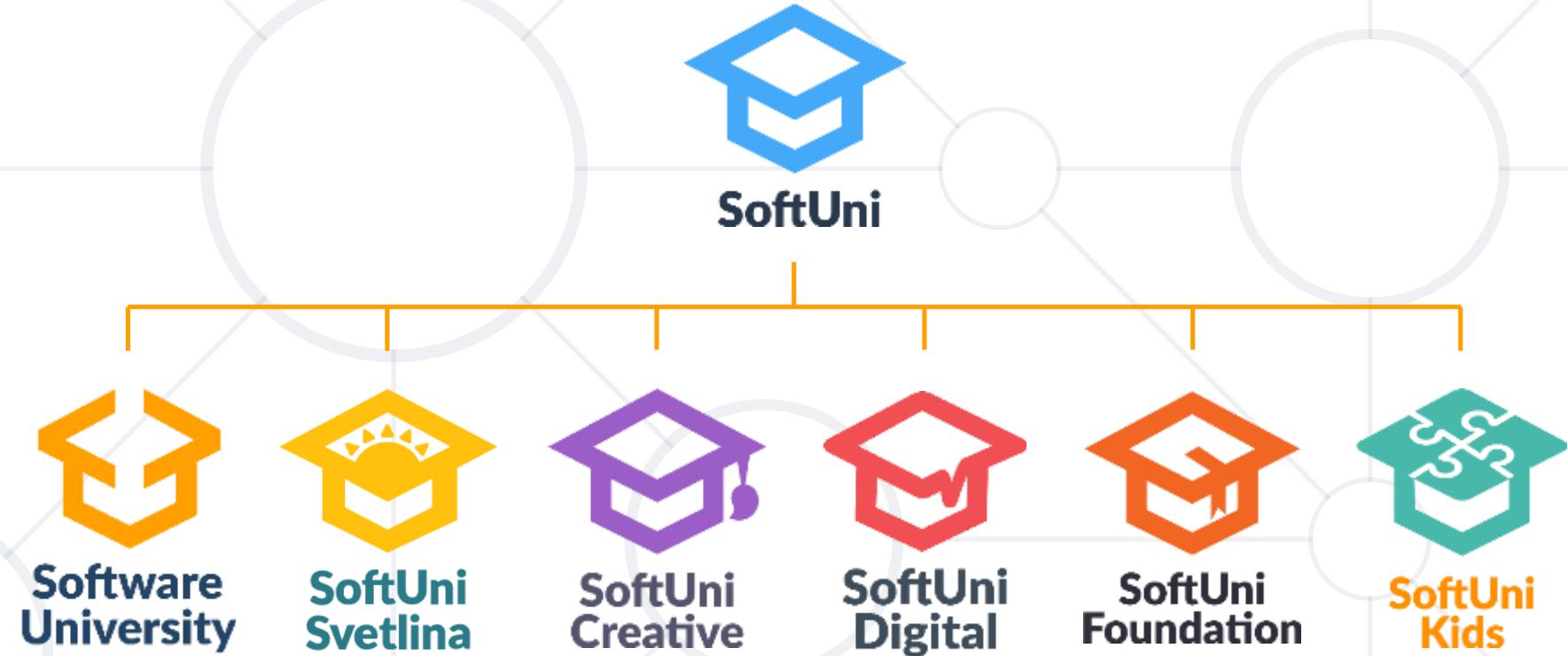
    cmd.Parameters.AddWithValue("@name", name);
    cmd.Parameters.AddWithValue("@desc", description);
    cmd.Parameters.AddWithValue("@start", startDate);

    cmd.ExecuteNonQuery();
}
```

- ADO.NET provides an interface between our apps and the database engine
- Different engines can be used with other data providers
- SQL **commands** must be **parametrized** to prevent malicious behavior



Questions?



SoftUni Diamond Partners



SCHWARZ



Coca-Cola HBC
Bulgaria



Postbank

Решения за твоето упре



**SOFTWARE
GROUP**

Bosch.**.IO**



SmartIT

POKERSTARS

CAREERS

AMBITIONED

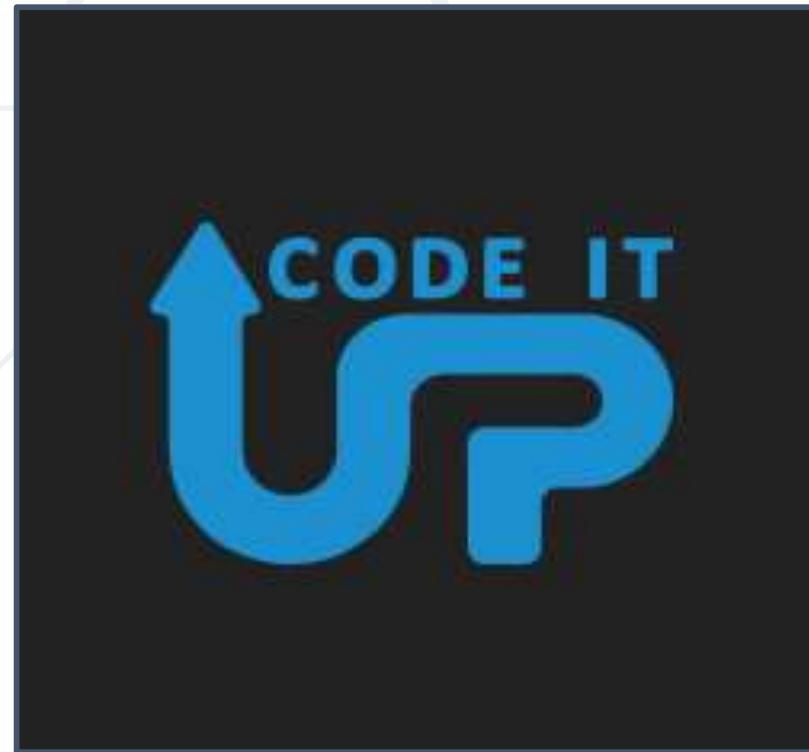
INDEAVR
Serving the high achievers

createX

**DRAFT
KINGS**

**SUPER
HOSTING
.BG**

Educational Partners



Trainings @ Software University (SoftUni)



- 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
- Software University Forums
 - forum.softuni.bg



Software
University

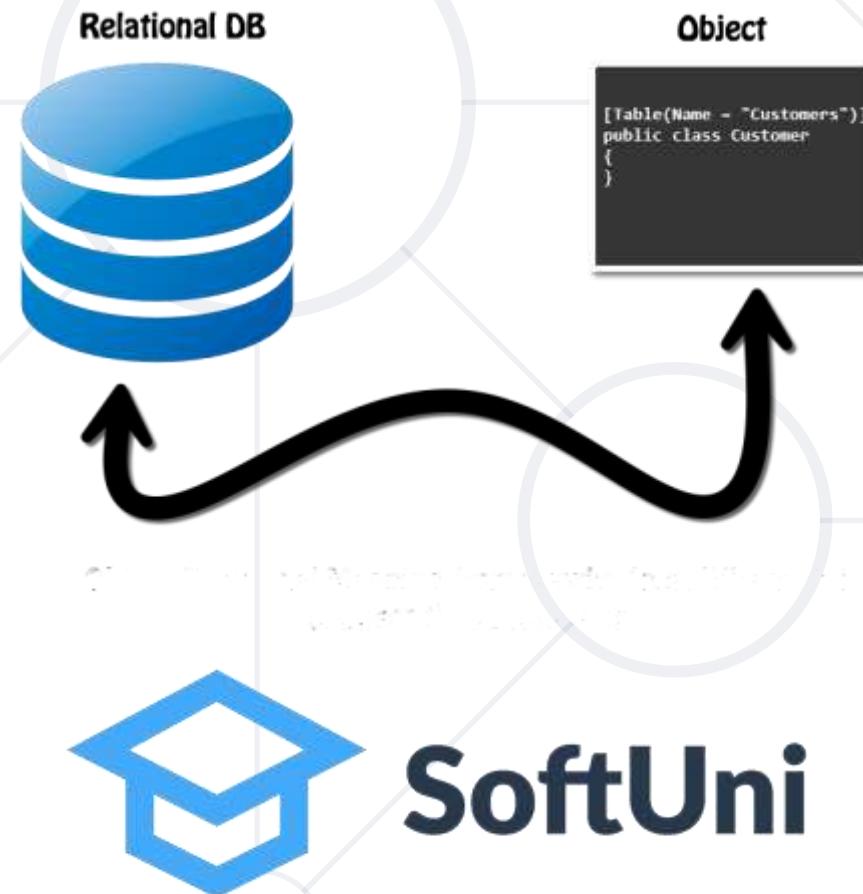


- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – <https://about.softuni.bg>
- © Software University – <https://softuni.bg>



ORM Fundamentals

The ORM Concept, Config, CRUD Operations



SoftUni Team

Technical Trainers



SoftUni

Software University

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

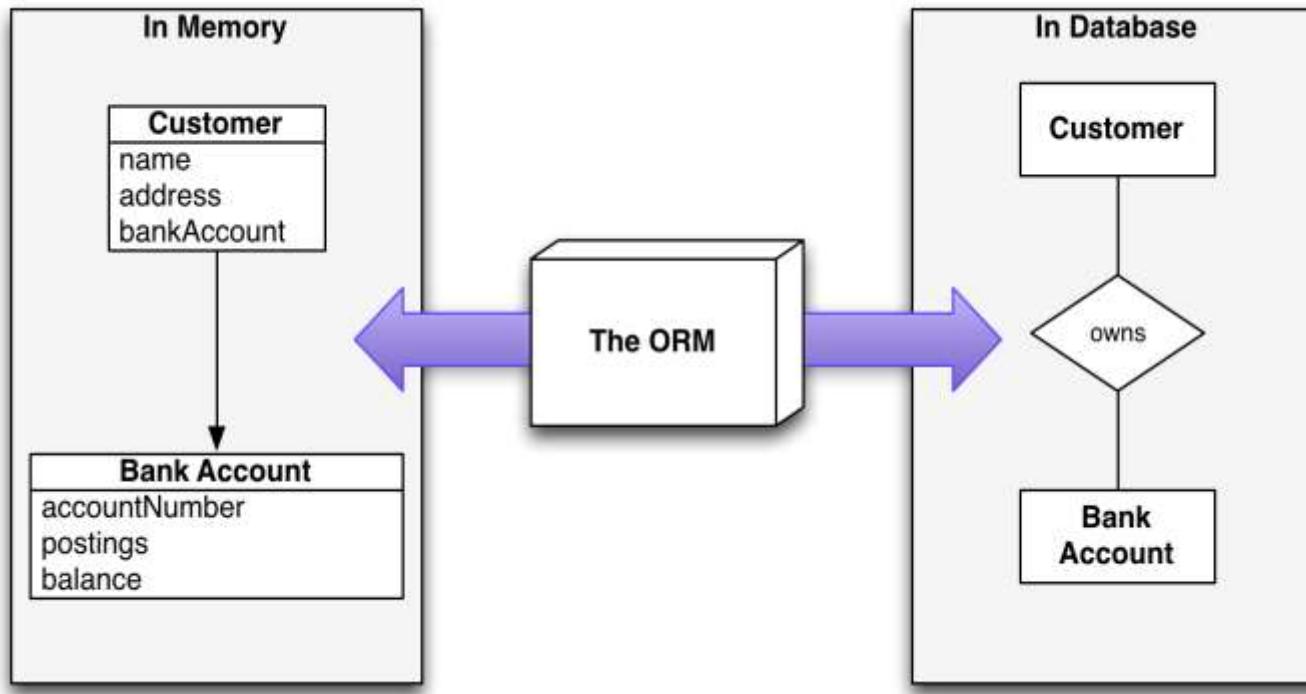
Table of Contents

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



sli.do

#csharp-db



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: Features

- **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    });
```

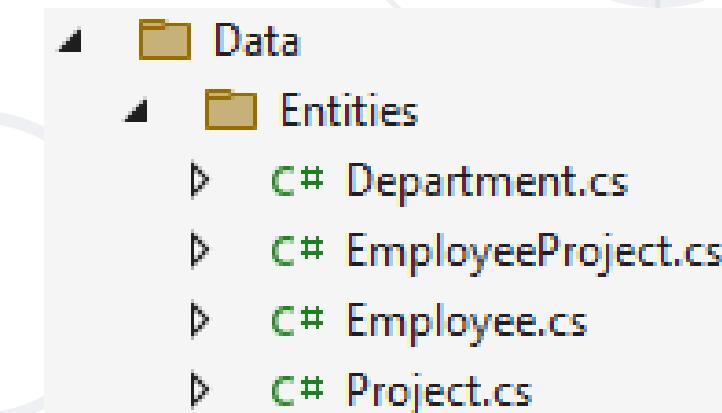
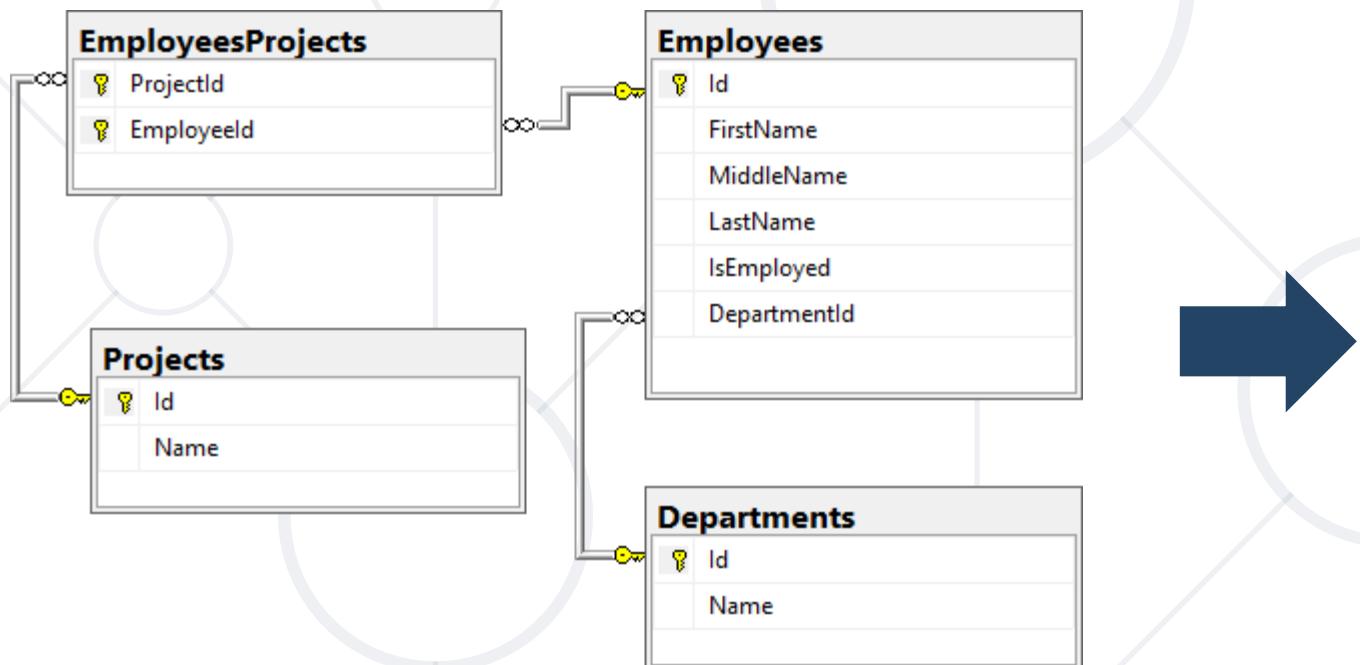


```
INSERT INTO Employees  
(FirstName, LastName, IsEmployed)  
VALUES  
( 'George ', 'Peterson', 1)
```

- **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)

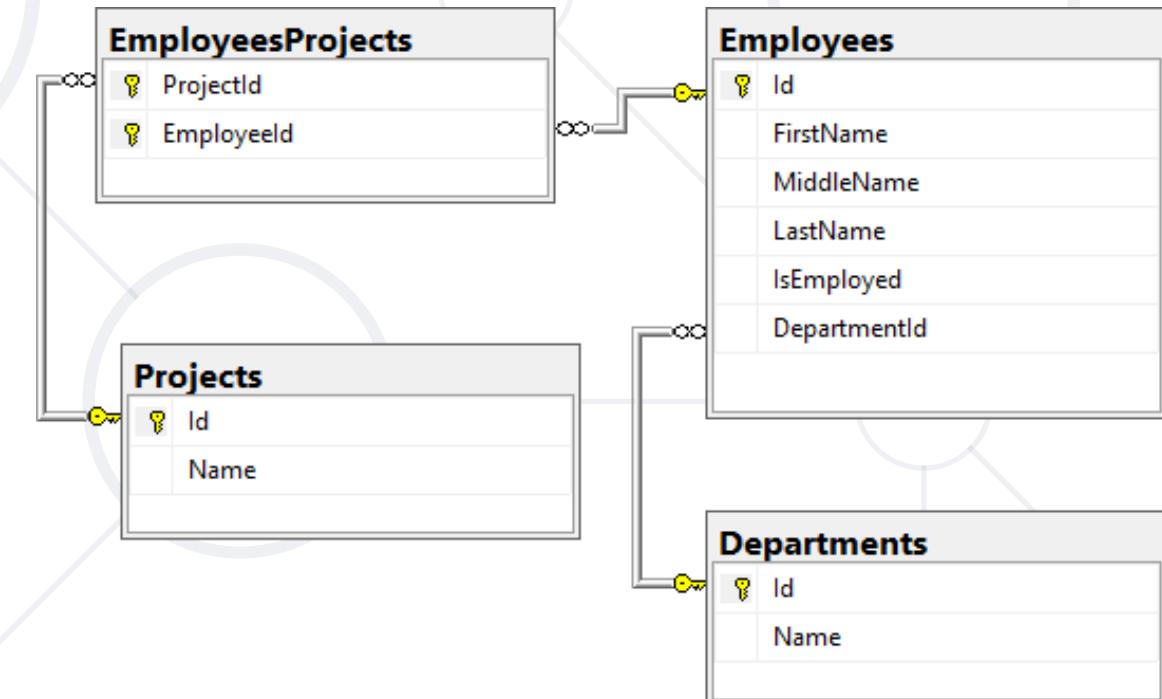
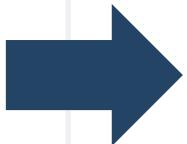
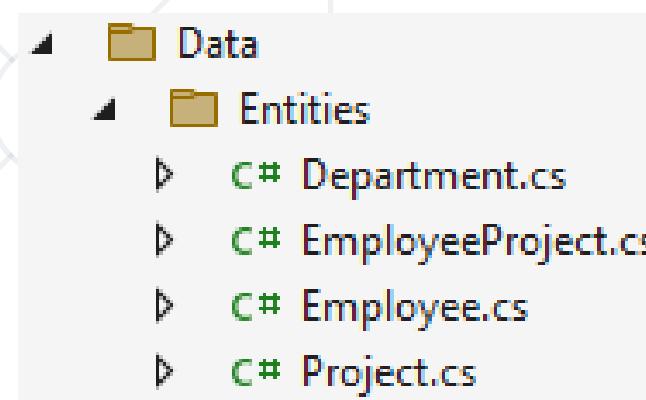
Database First Model

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



Code-First Model

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



ORM Advantages and Disadvantages

■ 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

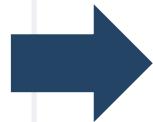
Data Holders

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; }
}
```

Entity Classes: Navigation Properties (1)

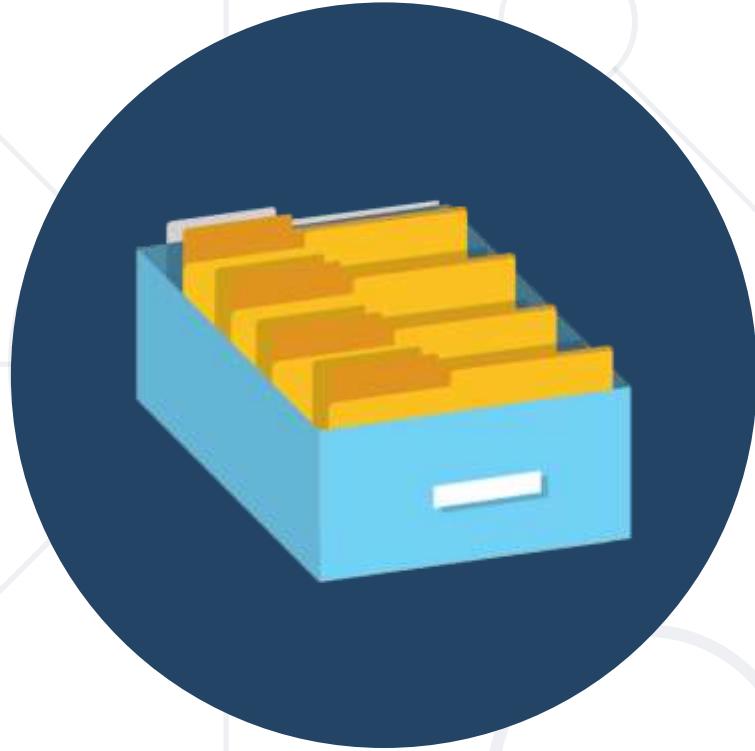
- Reference type properties
- Point to relevant object, connected by foreign key
- Set by the framework
- Example: Employee's Department

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

Entity Classes: Navigation Properties (2)

- 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**



DbSet<T> Features

- 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

Using DbContext Class

- 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; }**

Reading Data with LINQ Query (1)

- 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; }
}
```

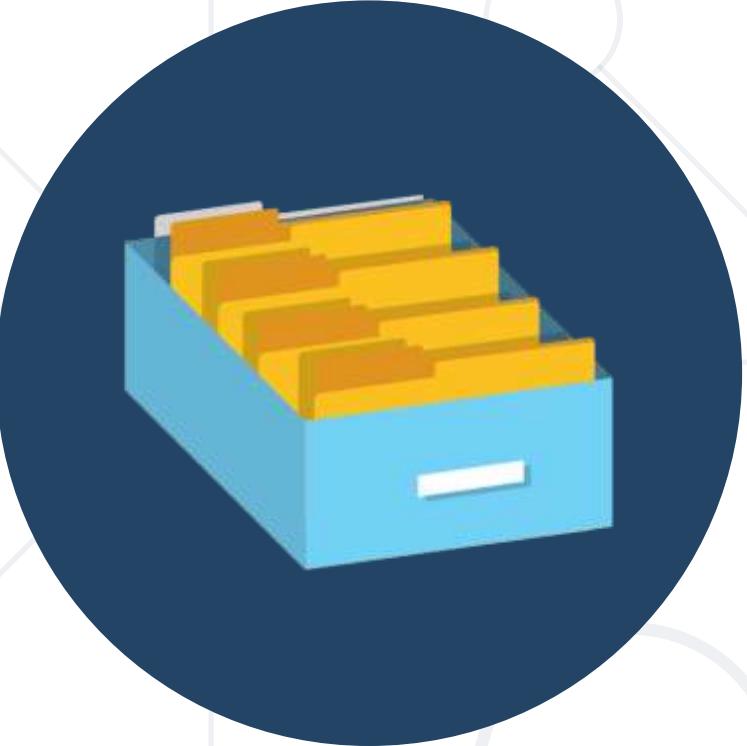
Reading Data with LINQ Query (2)

- 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);
```

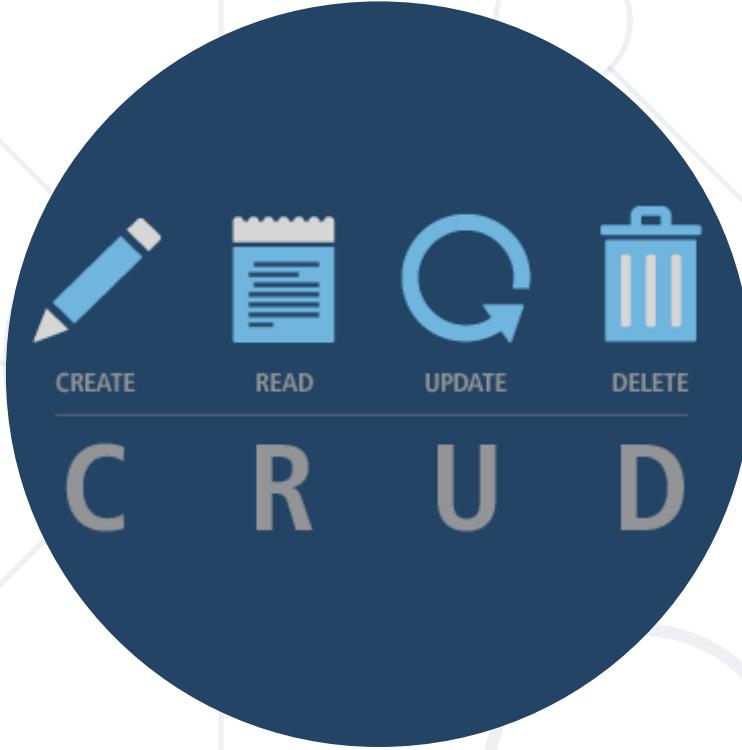


Change Tracking

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

Creating New Entities

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

```
var project = new Project()  
{  
    Name = "Judge System"  
};  
  
context.Projects.Add(project);  
context.SaveChanges();
```

Create a new
Project object

Add the object to the DbSet

Execute SQL statements

Updating Existing Entities

- **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

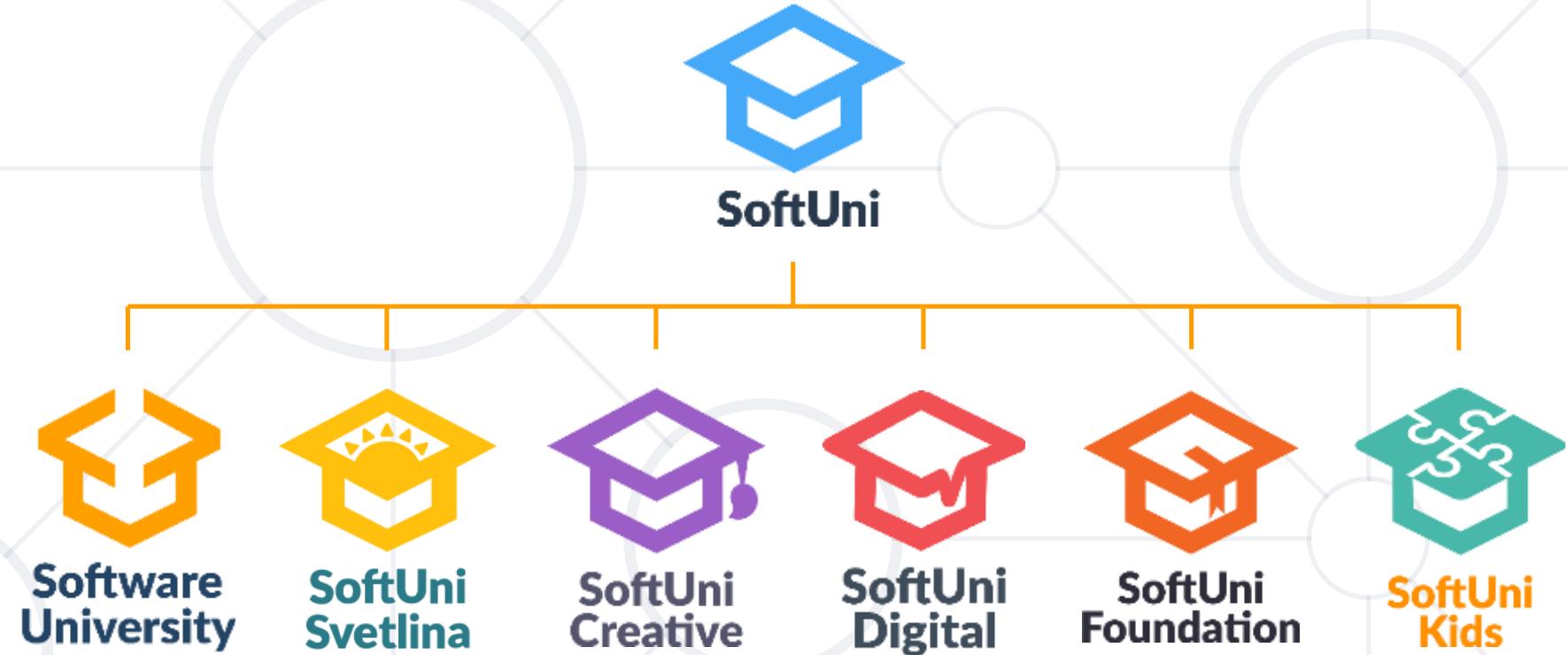
Execute the SQL
DELETE command

Summary

- **ORM frameworks** map database schema to objects in a programming language
- **LINQ** can be used to query the DB through the **DB Context**



Questions?



SoftUni Diamond Partners



SCHWARZ



Coca-Cola HBC
Bulgaria



Postbank

Решения за твоето упре



**SOFTWARE
GROUP**

Bosch.**.IO**



SmartIT



**PHAR
VISION**



CAREERS



INDEAVR
Serving the high achievers



**SUPER
HOSTING
.BG**

Educational Partners



Trainings @ Software University (SoftUni)



- 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
- Software University Forums
 - forum.softuni.bg



Software
University



- 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>



Entity Framework Core - Introduction

The ORM Concept



SoftUni Team

Technical Trainers



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

Table of Contents

- Entity Framework Core
- EF Core Components
- Reading Data
- CRUD Operations
- EF Core Configuration
 - Fluent API
- Database Migrations



Have a Question?



sli.do

#csharp-db



Entity Framework Core

Overview and Features

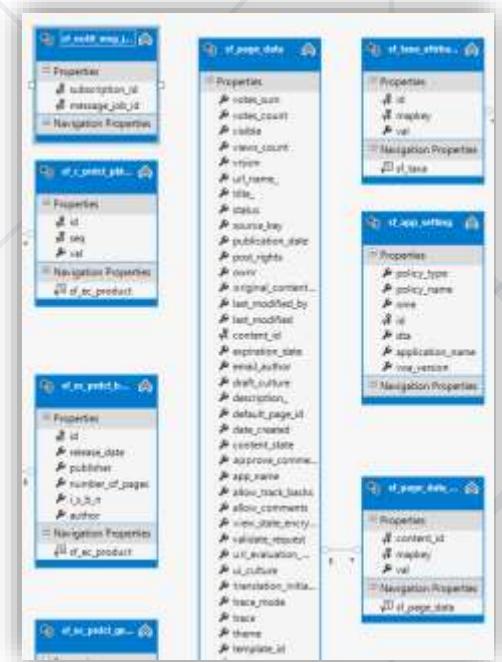
Entity Framework Core: Overview

- The standard **ORM framework** for **.NET** and **.NET Core**
- Provides LINQ-based data queries and **CRUD** operations
- Automatic **change tracking** of in-memory objects
- Works with many relational databases (with different providers)
- Open source with independent release cycle



EF Core: Basic Workflow (1)

1. Define the data model (**Code First or Scaffold from DB**)



2. Write & execute query over **IQueryable**

```
var toolName = "";

var snippetOptions = DefaultToolGroup
    .Tools
    .OfType<EditorListTool>()
    .Where(t =>
        t.Name == toolName &&
        t.Items != null &&
        t.Items.Any())
    .SelectMany(
        t, index) =>
        t.Items
        .Select(item =>
            new
            {
                text = item.Text,
                value = item.Value
            }));
if(snippetOptions.Any())
{
    options[toolName] = snippetOptions;
```

3. EF generates & executes an **SQL query** in the DB

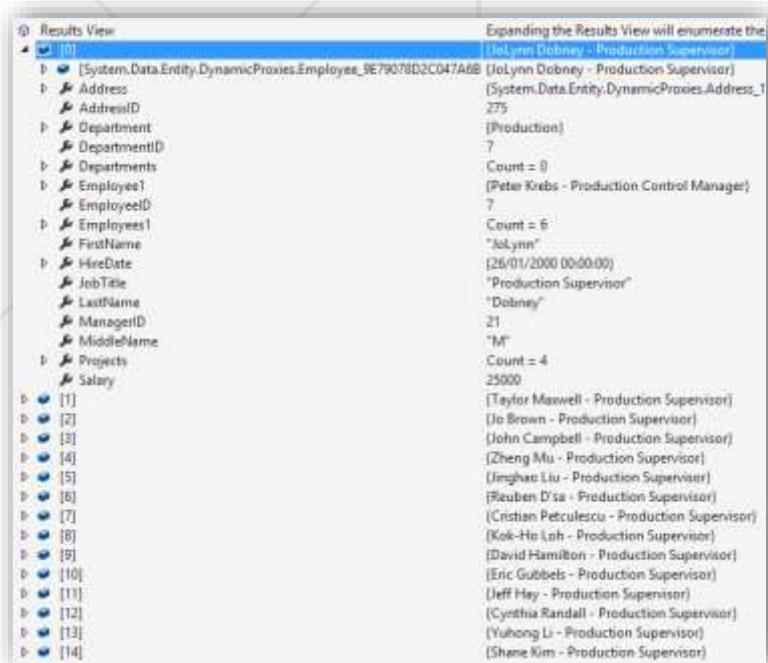
```
exec sp_executesql N'SELECT
[Filter2].[UserInCourseId] AS [UserInCourse]
,[Filter2].[UserId] AS [UserId],
,[Filter2].[CourseInstanceId] AS [CourseIns
,[Filter2].[FirstCourseGroupId] AS [FirstCou
,[Filter2].[SecondCourseGroupId] AS [SecondC
,[Filter2].[ThirdCourseGroupId] AS [ThirdCou
,[Filter2].[FourthCourseGroupId] AS [FourthC
,[Filter2].[FifthCourseGroupId] AS [FifthCou
,[Filter2].[IsLiveParticipant] AS [IsLivePar
,[Filter2].[Accommodation] AS [Accommodation
,[Filter2].[ExcellentResults] AS [ExcellentR
,[Filter2].[Result] AS [Result],
,[Filter2].[CanDoTestExam] AS [CanDoTestExam
,[Filter2].[CourseTestExamId] AS [CourseTest
,[Filter2].[TestExamPoints] AS [TestExamPoin
,[Filter2].[CanDoPracticalExam] AS [CanDoPra
,[Filter2].[CoursePracticalExamId] AS [Course
,[Filter2].[PracticalExamPoints] AS [Practic
,[Filter2].[AttendancesCount] AS [Attendance
,[Filter2].[HomeworkEvaluationPoints] AS [Ho
FROM (SELECT [Extent1].[UserInCourseId] A
AS [SecondCourseGroupId], [Extent1].[ThirdC
[IsLiveParticipant], [Extent1].[Accommodati
[CourseTestExamId], [Extent1].[TestExamPoin
[PracticalExamPoints], [Extent1].[Attendance
FROM [courses].[UsersInCourses] AS
INNER JOIN [courses].[CoursePractic
WHERE ( EXISTS (SELECT
1 AS [c1]
FROM [courses].[CoursePract
WHERE [Extent1].[UserInCour
)) AND ([Extent2].[AllowExamFileSav
INNER JOIN [courses].[CoursePracticalExams]
WHERE ([Filter2].[UserId] = @p_linq_0) AN
```

EF Core: Basic Workflow (2)

4. EF transforms
the query
results into
.NET objects

5. Modify data with C# code and call **Save Changes()**

6. Entity Framework generates & executes SQL command to modify the DB



```
private void ChangeBlogPostName(int id,
    string newName)
{
    var db = new Context();
    var post = db.Posts
        .FirstOrDefault(x => x.Id == id);

    if(post == null)
    {
        throw new ArgumentNullException(
            "Item with that id was not found");
    }

    post.Name = newName;
    db.SaveChanges();
}
```

```
exec sp_executesql N'SET NOCOUNT ON;
DELETE FROM [Categories]
WHERE [CategoryID] = @p0;
SELECT @@ROWCOUNT;
UPDATE [categories] SET [categoryName] = @p1
WHERE [CategoryID] = @p2;
SELECT @@ROWCOUNT;
INSERT INTO [Categories] ([CategoryID], [categoryName])
VALUES (@p3, @p4),
(@p5, @p6);
```

Entity Framework Core: Setup

- To add **EF Core support** to a project in **Visual Studio**
 - Install it from **NuGet** using Visual Studio or **dotnet CLI**

```
dotnet add package Microsoft.EntityFrameworkCore
```
 - EF Core is modular – any **data providers** must be installed too

Microsoft.EntityFrameworkCore.SqlServer

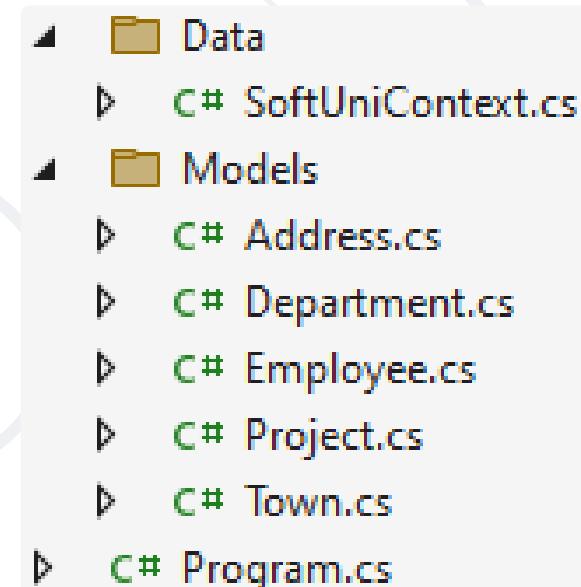
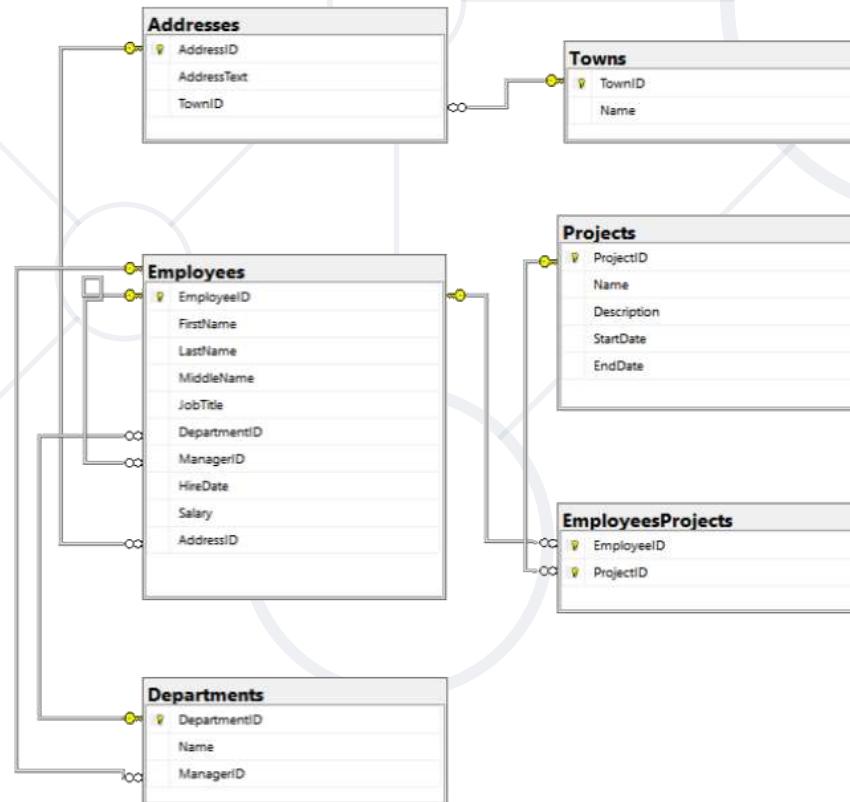
- To use the Entity Framework Core CLI tools

```
dotnet tool install --global dotnet-ef
```

```
dotnet add package Microsoft.EntityFrameworkCore.Design
```

Database First Model

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



Database First Model: Setup

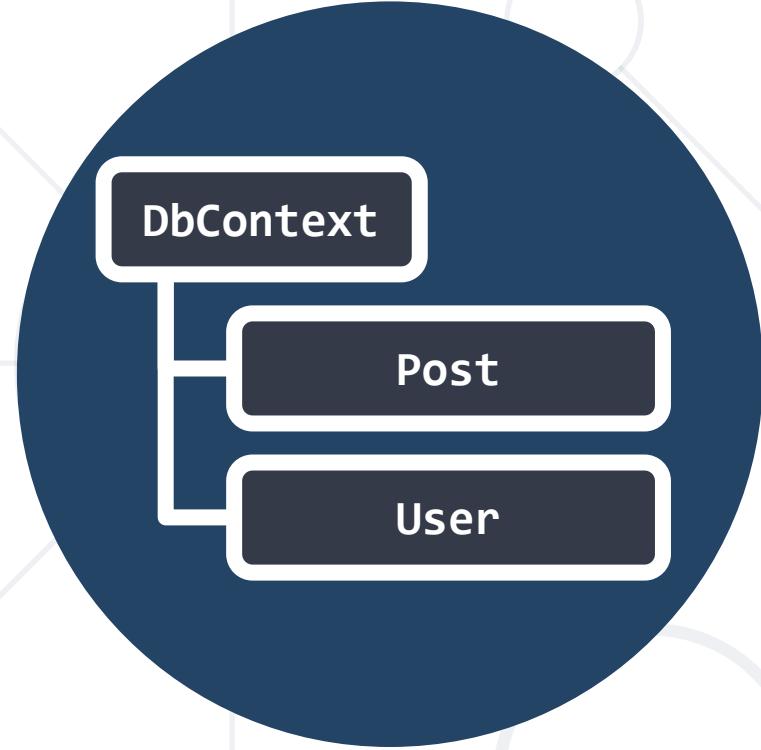
- Scaffolding **DbContext** from DB with **EF Core CLI Tools**

```
dotnet ef dbcontext scaffold "Server=...;Database=...;Integrated  
Security=true" Microsoft.EntityFrameworkCore.SqlServer -o Models
```

- To update with the latest database changes, use the **-f** flag
 - To use attributes for configuring the model use the **-d** flag

```
dotnet ef dbcontext scaffold "..." Microsoft... -o Models -f -d
```

- Scaffolding requires the following NuGet packages installed
 - **Microsoft.EntityFrameworkCore.SqlServer**
 - **Microsoft.EntityFrameworkCore.Design**



EF Core Components

Overview of System Objects

Domain Classes (Models)

- Bunch of normal C# classes (POCO)
 - May contain **navigation properties** for table relationships

```
public class PostAnswer
{
    public int Id { get; set; }
    public string Content { get; set; }
    public int PostId { get; set; }
    public Post Post { get; set; }
}
```



The diagram shows three callout boxes pointing to specific parts of the code. The first box, labeled 'Primary key', points to the `Id` property. The second box, labeled 'Foreign key', points to the `PostId` property. The third box, labeled 'Navigation property', points to the `Post` property.

- Recommended to be in a **separate class library**

DbSet Type

- Maps a **collection of entities** from a **table**
- Set operations: **Add, Attach, Remove, Find**
- DbContext** contains multiple **DbSet<T>** properties

```
public class DbSet<TEntity> :  
    System.Data.Entity.Infrastructure.DbQuery<TEntity>  
    where TEntity : class  
Member of System.Data.Entity
```

```
public DbSet<Post> Posts { get; set; }
```

The DbContext Class

- Usually named after the database, e.g., **BlogDbContext**, **ForumDbContext**
- Inherits from **DbContext**
- Manages model classes using **DbSet<T>** type
- Implements **identity tracking**, **change tracking**
- Provides **API** for **CRUD** operations and **LINQ-based** data access
- Recommended to be in a separate class library
 - Don't forget to reference the EF Core library + any providers
- Use several **DbContext** if you have too much models

Defining DbContext Class - Example

```
using Microsoft.EntityFrameworkCore;
using CodeFirst.Data.Models; Models Namespace
public class ForumDbContext : DbContext
{
    public DbSet<Category> Categories { get; set; }
    public DbSet<Post> Posts { get; set; }
    public DbSet<User> Users { get; set; }
}
```

EF Reference



Reading Data

Querying the DB Using Entity Framework

The DbContext Class

- **DbContext** provides
 - CRUD Operations
 - A way to **access entities**
 - Methods for **creating** new entities (**Add()** method)
 - Ability to **manipulate database data by modifying objects**
 - Easily navigate through **table relations**
 - Executing **LINQ queries** as native **SQL queries**
 - Managing database **creation/deletion/migration**

Using DbContext Class

- First create instance of the **DbContext**

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

- In the constructor you can pass a database connection string
- **DbContext** properties
 - **Database** - **EnsureCreated/Deleted** methods, DB Connection
 - **ChangeTracker** - Holds info about the **automatic change tracker**
 - All entity classes (tables) are listed as properties
 - e.g., **DbSet<Employee> Employees { get; set; }**

Reading Data with LINQ Query (1)

- Executing **LINQ-to-SQL** query over EF entity

```
using (var context = new SoftUniEntities())
{
    var employees = context.Employees
        .Where(e => e.JobTitle == "Design Engineer")
        .ToArray();
}
```

EF translates this
to an SQL query

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

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

Reading Data with LINQ Query (2)

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

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

ToList() materializes the query

- Find element by **ID**

```
using (var context = new SoftUniEntities())
{
    var project = context.Projects.Find(2);
    Console.WriteLine(project.Name);
}
```

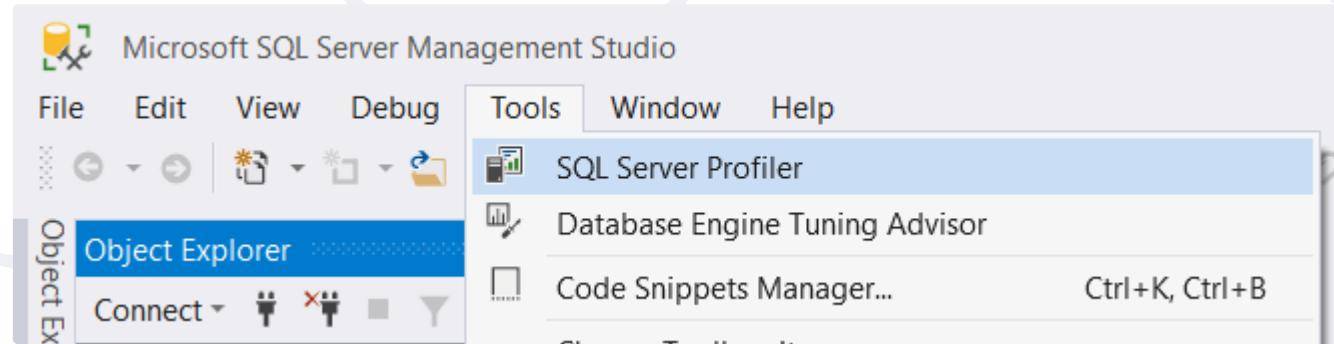
- **Where()**
 - Searches by given condition
- **First()/Last() / FirstOrDefault() / LastOrDefault()**
 - Gets the **first/last** element which matches the condition
 - Throws **InvalidOperationException** without **OrDefault**
- **Select()**
 - Projects (conversion) collection to another type
- **OrderBy() / ThenBy() / OrderByDescending()**
 - Orders a collection

LINQ: Simple Operations (2)

- **Any()**
 - Checks if any element matches a condition
- **All()**
 - Checks if all elements match a condition
- **Distinct()**
 - Returns only unique elements
- **Skip() / Take()**
 - Skips or takes X number of elements

Logging the Native SQL Queries

- Queries sent to SQL Server can be **monitored** with the SQL Server Profiler
 - Included with the **SQL Server** installation



- Queries can be gotten using the built-in **ToQueryString()** method

```
db.Courses.Where(x => x.Title == "EF Core").ToQueryString()
```



CRUD

CRUD Operations

With Entity Framework

Creating New Data

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

```
var project = new Project()
{
    Name = "Judge System",
    StartDate = new DateTime(2023, 1, 26),
};

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

Create a new
Project
object

Add the object to the **DbSet**

Execute SQL statements

Cascading Inserts

- We can also add cascading entities to the database

```
Employee employee = new Employee();
employee.FirstName = "John";
employee.LastName = "Doe";
employee.Projects.Add(new Project { Name = "SoftUni Conf" } );
softUniEntities.Employees.Add(employee);
softUniEntities.SaveChanges();
```

- The **Project** will be added when the **Employee** entity (employee) is inserted to the database

Updating Existing Data

- **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

```
Employees employee =  
    softUniEntities.Employees.First();  
employee.FirstName = "Alex";  
context.SaveChanges();
```

SELECT the
first order

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

```
Employees employee =  
softUniEntities.Employees.First();  
  
softUniEntities.Employees.Remove(employee);  
  
softUniEntities.SaveChanges();
```

Mark the entity for deleting
at the next save

Execute the SQL DELETE
command



EF Core Configuration

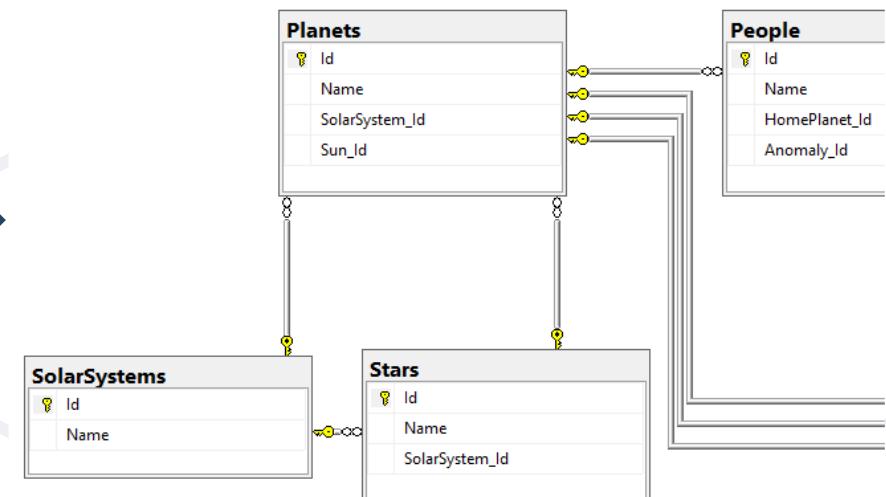
NuGet Packages, Configuration

What is the Code First Model?

- **Code First** means to write the .NET classes and let EF Core create the **database** from the **mappings**



```
C# Planet.cs
  ↘ Planet
    ↗ Planet()
    ↗ Id : int
    ↗ Name : string
    ↗ Sun : Star
    ↗ SolarSystem : SolarSystem
    ↗ OriginAnomalies : ICollection<Anomaly>
C# SolarSystem.cs
  ↘ SolarSystem
    ↗ Id : int
    ↗ Name : string
C# Star.cs
  ↘ Star
    ↗ Id : int
    ↗ Name : string
    ↗ SolarSystem : SolarSystem
```



Why Use Code First?

- Write code **without** having to define **mappings** in XML or **create** database **tables**
- Define objects in **C# format**
- Enables database persistence with no configuration
- Changes to code can be **reflected** (migrated) in the schema
- **Data Annotations** or **Fluent API** describe properties
 - **Key**, **Required**, **MinLength**, etc.

Code First with EF Core: Setup

- To add EF Core support to a project in Visual Studio

- Install it from **Package Manager Console**

```
Install-Package Microsoft.EntityFrameworkCore
```

- Or using **.NET Core CLI**

```
dotnet add package Microsoft.EntityFrameworkCore
```

- EF Core is modular – any **data providers** must be installed too

```
Microsoft.EntityFrameworkCore.SqlServer
```

How to Connect to SQL Server?

- One way to connect is to create a **Configuration** class with your connection string

```
public static class Configuration
{
    public const string ConnectionString = "Server=.;Database=...;";
```

- Then add the connection string in the **OnConfiguring** method in the **DbContext** class

```
protected override void OnConfiguring(DbContextOptionsBuilder builder)
{
    if (!builder.IsConfigured)
        builder.UseSqlServer(Configuration.ConnectionString);
}
```

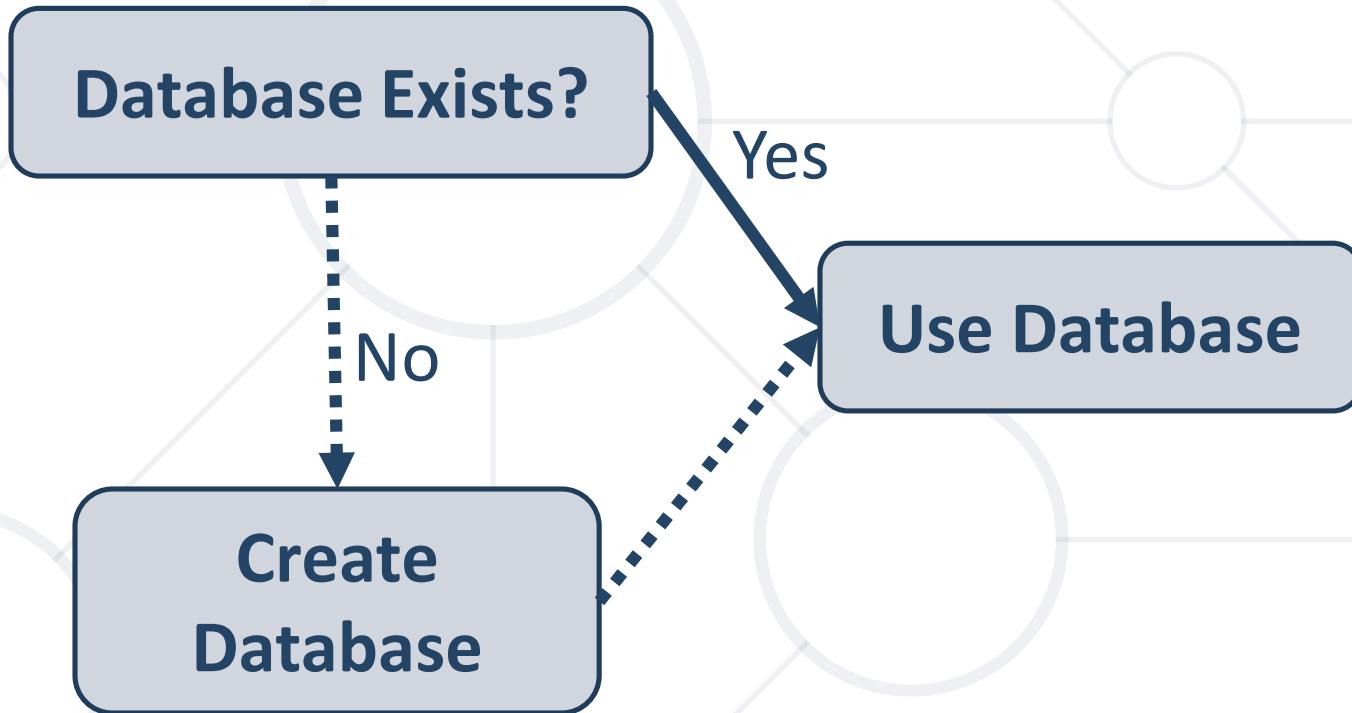
- The **OnModelCreating** method let us use the Fluent API to describe our **table relations** to **EF Core**

```
protected override void OnModelCreating(ModelBuilder builder)
{
    builder.Entity<Category>()
        .HasMany(c => c.Posts)
        .WithOne(p => p.Category);

    builder.Entity<Post>()
        .HasMany(p => p.Replies)
        .WithOne(r => r.Post);

    builder.Entity<User>()
        .HasMany(u => u.Posts)
        .WithOne(p => p.Author);
}
```

Database Connection Workflow

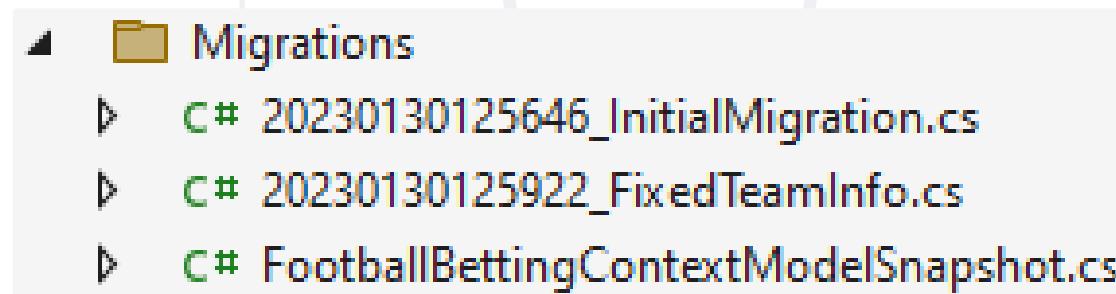




Database Migrations

What Are Database Migrations?

- Updating database schema **without losing data**
 - Adding/dropping tables, columns, etc.
- Migrations in EF Core keep their **history**
 - Entity Classes, DB Context versions are all **preserved**
- **Automatically** generated



Migrations in EF Core

- To use migrations in EF Core, we use the **dotnet ef migrations add** command from the EF CLI Tools

```
dotnet ef migrations add {MigrationName}
```

- To undo a migration, we use **migrations remove**

```
dotnet ef migrations remove {MigrationName}
```

- Commit changes to the database

```
dotnet ef database update
```

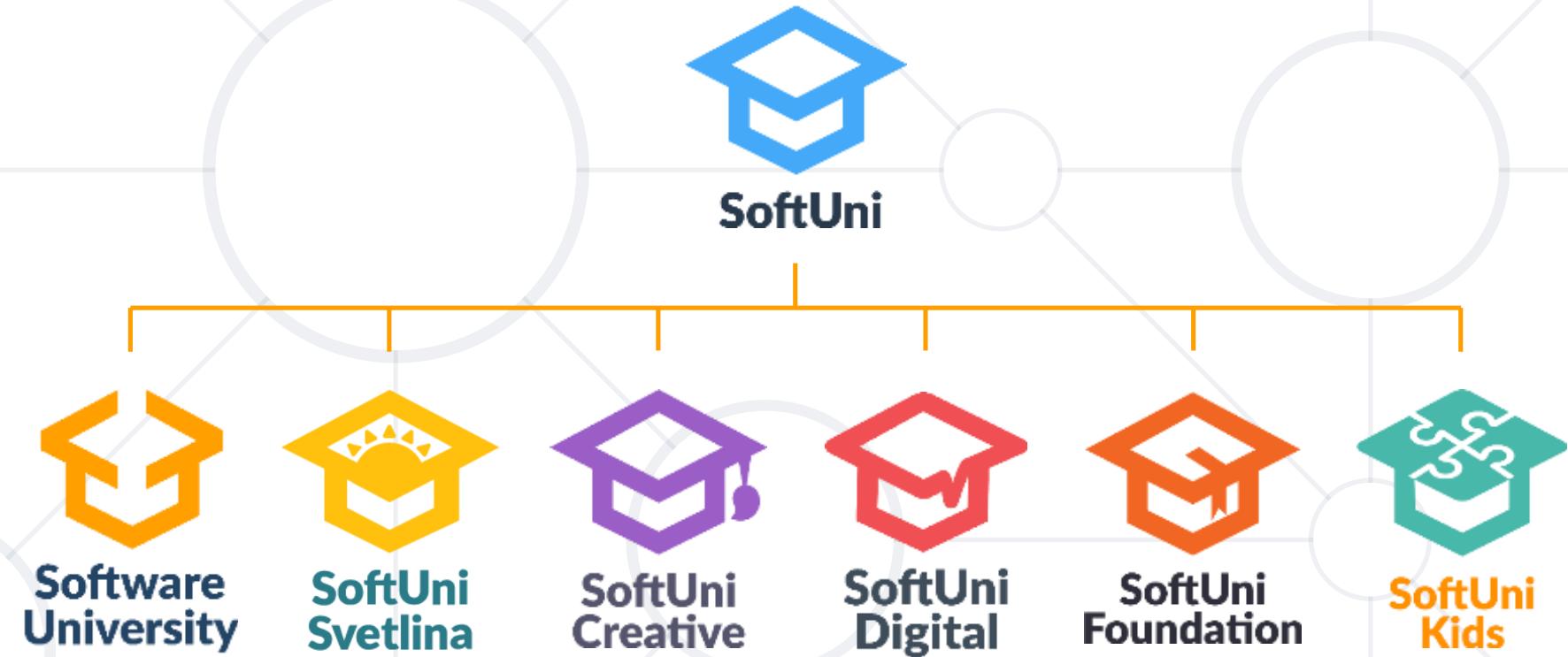
```
db.Database.Migrate()
```

Summary

- ORM frameworks maps database schema to objects in a programming language
- Entity Framework Core is the standard .NET ORM
- LINQ can be used to query the DB through the DB context



Questions?



SoftUni Diamond Partners



SCHWARZ



Coca-Cola HBC
Bulgaria



Postbank

Решения за твоето упре



Bosch..IO****



SmartIT

POKERSTARS

CAREERS

AMBITIONED

INDEAVR
Serving the high achievers

createX

DRAFT KINGS

SUPER HOSTING .BG

Educational Partners



Trainings @ Software University (SoftUni)



- 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
- Software University Forums
 - forum.softuni.bg



Software
University



- 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>



Entity Relations

Customizing Entity Models



SoftUni

SoftUni Team

Technical Trainers

 Software University



Software University

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

Table of Contents

- Object Composition
- Fluent API
- Attributes
- Table Relationships
 - One-to-One
 - One-to-Many
 - Many-to-Many

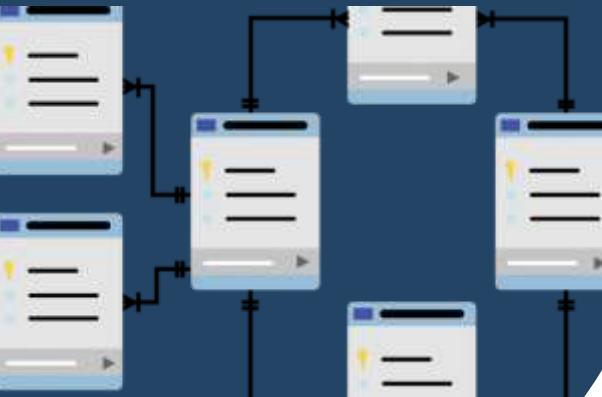


Have a Question?



sli.do

#csharp-db

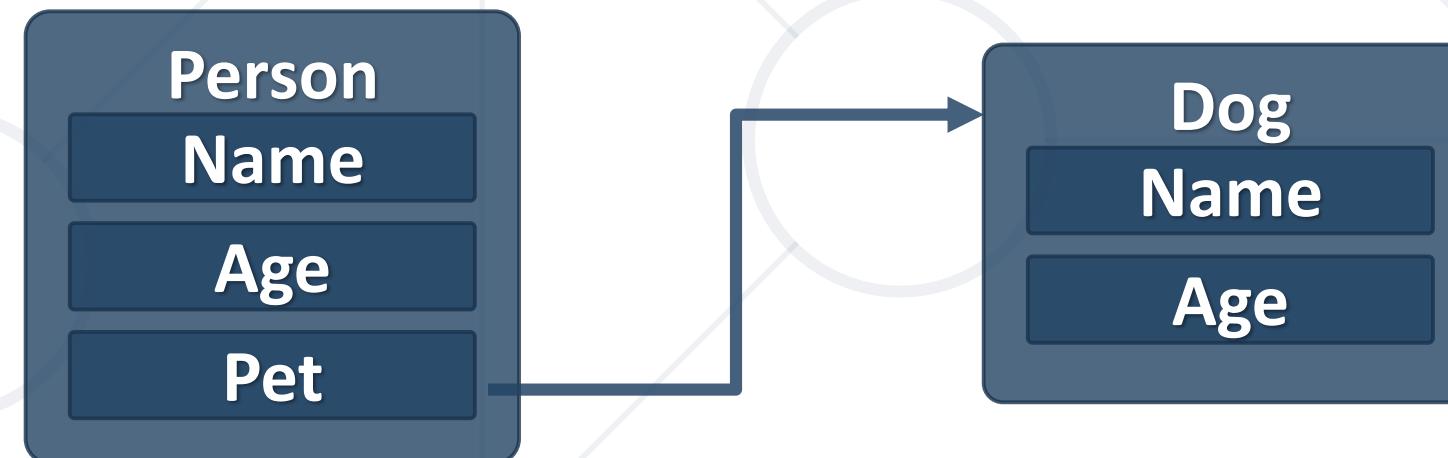


Object Composition

Describing Database Relationships

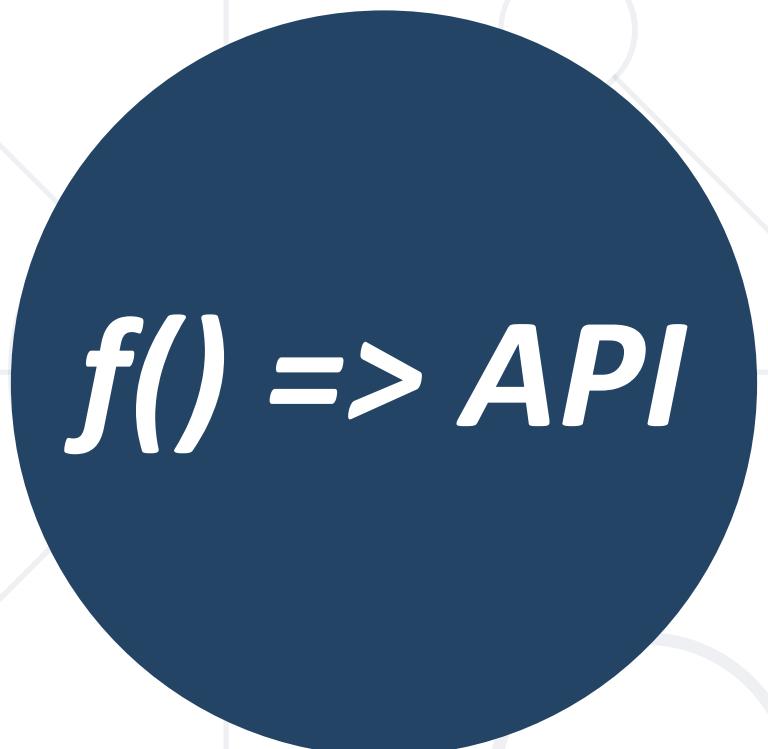
Object Composition

- Object composition denotes a "**has-a**" relationship
 - e.g., the **car** has an **engine**
- Defined in C# by one object having a property that is a reference to another



Navigation Properties

- Navigation properties create a **relationship** between entities
 - Either an **Entity Reference** (one-to-one-or-zero) or an **ICollection** (one-to-many or many-to-many)
- They provide **fast querying** of related records
- Can be **modified** by **directly** setting the reference



f() => API

Fluent API

Working with Model Builder

- **Code First** maps your POCO (Plain Old CLR Objects) classes to tables using a **set of conventions**
 - e.g., property named "**Id**" maps to the **Primary Key**
- Can be customized using **annotations** and the **Fluent API**
- Fluent API (Model Builder) allows **full control** over DB mappings
 - Custom names of objects (columns, tables, etc.) in the DB
 - Validation and data types
 - Define complicated entity relationships

Working with Fluent API

- Custom mappings are placed inside the **OnModelCreating** method of the DB context class

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Student>()
        .HasKey(s => s.StudentKey);
}
```

Fluent API: Renaming DB Objects

- Specifying Custom Table name

```
modelBuilder.Entity<Order>()
    .ToTable("OrderRef", "Admin");
```

Optional schema
name

- Custom Column name/DB Type

```
modelBuilder.Entity<Student>()
    .Property(s => s.Name)
    .HasColumnName("StudentName")
    .HasColumnType("varchar");
```

Fluent API: Column Attributes

- Explicitly set Primary Key

```
modelBuilder  
    .Entity<Student>()  
    .HasKey("StudentKey");
```

- Other column attributes

```
modelBuilder.Entity<Person>()  
    .Property(p => p.FirstName)  
    .IsRequired()  
    .HasMaxLength(50)
```

```
modelBuilder.Entity<Post>()  
    .Property(p => p.LastUpdated)  
    .ValueGeneratedOnAddOrUpdate()
```

Fluent API: Miscellaneous Config

- Do not include property in DB (e.g., business logic properties)

```
modelBuilder  
    .Entity<Department>()  
    .Ignore(d => d.Budget);
```

- Disabling cascade delete
 - If a FK property is non-nullable, cascade delete is **on by default**

```
modelBuilder.Entity<Course>()  
    .HasRequired(t => t.Department)  
    .WithMany(t => t.Courses)  
    .HasForeignKey(d => d.DepartmentID)  
    .OnDelete(DeleteBehavior.Restrict);
```

Throws exception on delete

Specialized Configuration Classes

- Mappings can be placed in entity-specific classes

```
public class StudentConfiguration  
    : IEntityConfiguration<Student>  
{  
    public void Configure(EntityTypeBuilder<Student> builder)  
    {  
        builder.HasKey(c => c.StudentKey);  
    }  
}
```

Specify target model

- Include in **OnModelCreating**

```
builder.ApplyConfiguration(new StudentConfiguration());
```



Attributes

Custom Entity Framework Behavior

Attributes

- EF Code First provides a set of **DataAnnotation** attributes
 - You can override default Entity Framework behavior
- Access nullability and size of fields

```
using System.ComponentModel.DataAnnotations;
```
- Access schema customizations

```
using System.ComponentModel.DataAnnotations.Schema;
```
- For a full set of configuration options you need the **Fluent API**

Key Attribute

- **[Key]** – explicitly specify **primary key**
 - When your PK column doesn't have an "**Id**" or "**<TypeName>Id**" suffix

```
[Key]  
public int StudentKey { get; set; }
```

- **Composite key** is defined using **Fluent API**

```
builder.Entity<Car>()  
    .HasKey(c => new { c.State, c.LicensePlate });
```

- **[PrimaryKey]** available **only** in EF7

ForeignKey Attribute

- **[ForeignKey]** – explicitly **link** navigation property and foreign key property within the same class
- Works in **either direction** (FK to navigation property or navigation property to FK)
 - **[ForeignKey(NavigationPropertyName)]** – on the foreign key scalar property in the dependent entity
 - **[ForeignKey(ForeignKeyPropertyName)]** – on the related reference navigation property in the dependent entity
 - **[ForeignKey(ForeignKeyPropertyName)]** – on the navigation property in the principal entity

ForeignKey Attribute – Example

- `[ForeignKey(NavigationPropertyName)]`
- `[ForeignKey(ForeignKeyPropertyName)]`

```
public class Order
{
    ...
    [ForeignKey(nameof(Client))]
    public int ClientId { get; set; }
    public Client Client { get; set; }
}
```

```
public class Order
{
    ...
    public int ClientId { get; set; }
    [ForeignKey(nameof(ClientId))]
    public Client Client { get; set; }
}
```

```
public class Client
{
    ...
    public ICollection<Order> Orders { get; set; }
}
```



Renaming Objects (1)

- **Table** – manually specify the name of the table in the DB

```
[Table("StudentMaster")]
public class Student
{
    ...
}
```

```
[Table("StudentMaster", Schema = "Admin")]
public class Student
{
    ...
}
```

Renaming Objects (2)

- **Column** – manually specify the name of the column in the DB
 - You can also specify order and explicit data type

```
public class Student  
{  
    ...  
    [Column("StudentName", Order = 2, TypeName="varchar(50)")]  
    public string Name { get; set; }  
}
```

Optional parameters

- **Required** – mark a nullable property as **NOT NULL** in the DB
 - Will throw an exception if not set to a value
 - Non-nullable types (e.g., `int`) will **not throw** an exception (will be set to language-specific default value)
- **MinLength** – specifies min length of a string (client validation)
- **MaxLength / StringLength** – specifies max length of a string (both client and DB validation)
- **Range** – set lower and/or upper limits of numeric property (client validation)

Other Attributes

- **Index** – create index for column(s)
 - Primary key will always have an index

```
[Index(nameof(Url))]  
public class Student {  
    public string Url { get; set; }  
}
```

- **NotMapped** – property will not be mapped to a column
 - For business logic properties

```
[NotMapped]  
public string FullName => this.FirstName + this.LastName;
```



Table Relationships

Expressed As Properties and Attributes

One-to-Zero-or-One

- Expressed in SQL Server as a shared primary key
- Relationship direction must be explicitly specified with a **ForeignKey** attribute
- **ForeignKey** is placed above the key property and contains the **name** of the navigation property and vice versa



One-to-Zero-or-One: Implementation (1)

- Using the **ForeignKey** Attribute

```
public class Student
{
    [Key]
    public int Id { get; set; }
    public string Name { get; set; }
    [ForeignKey("Address")]
    public int AddressId { get; set; }
    public Address Address { get; set; }
}
```

Attributes

One-to-Zero-or-One: Implementation (2)

- Using the **ForeignKey** Attribute

```
public class Address
{
    public int Id { get; set; }
    public string Text { get; set; }
    [ForeignKey(nameof(Student))]
    public int StudentId { get; set; }
    public Student Student { get; set; }
}
```

One-to-Zero-or-One: Fluent API

- **HasOne → WithOne**

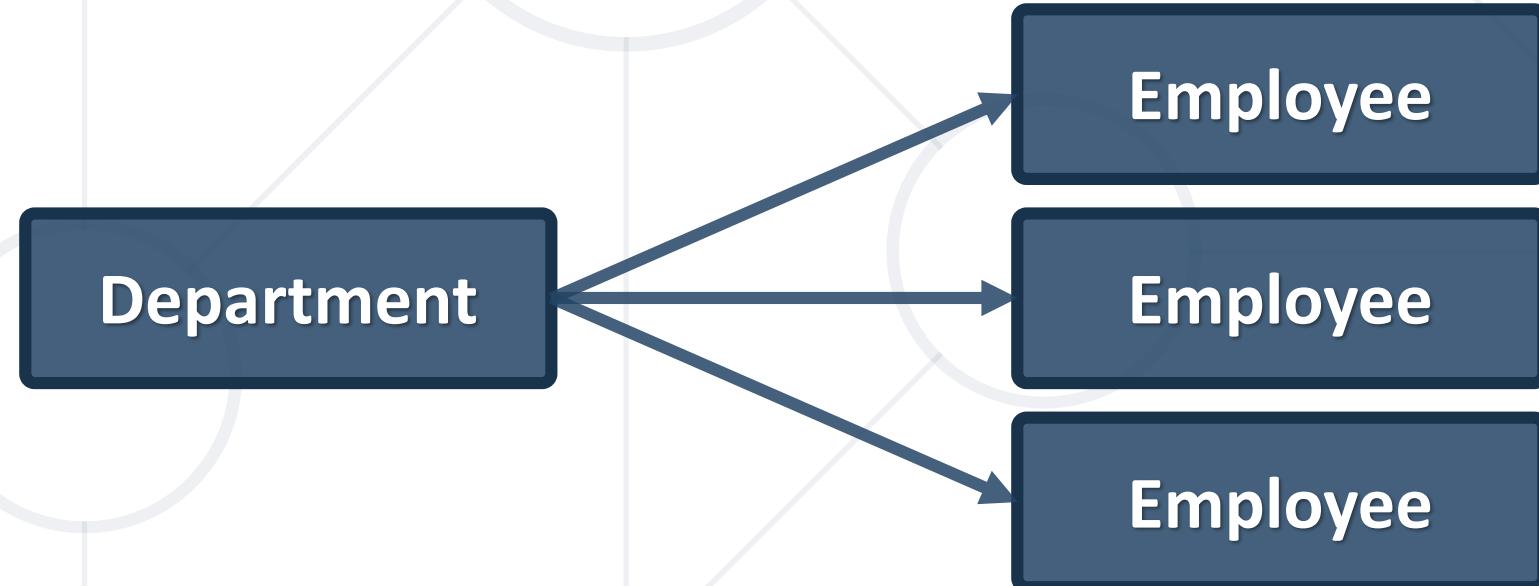
```
modelBuilder.Entity<Address>()
    .HasOne(a => a.Student)
    .WithOne(s => s.Address)
    .HasForeignKey<Student>(a => a.StudentId);
```

Address contains FK
to Student

- If **StudentId** property is **nullable (int?)**, relation becomes **One-To-Zero-Or-One**

One-to-Many

- Most common type of relationship
- Implemented with a **collection** inside the **parent entity**
 - The collection should be **initialized** in the **constructor!**



One-to-Many: Implementation (1)

- One department has many employees

```
public class Department
{
    public int Id { get; set; }
    public string Name { get; set; }

    public ICollection<Employee> Employees { get; set; }
}
```

One-to-Many: Implementation (2)

- Each employee has one department

```
public class Employee
{
    public int Id { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }

    public int DepartmentId { get; set; }
    public Department Department { get; set; }
}
```

One-to-Many: Fluent API

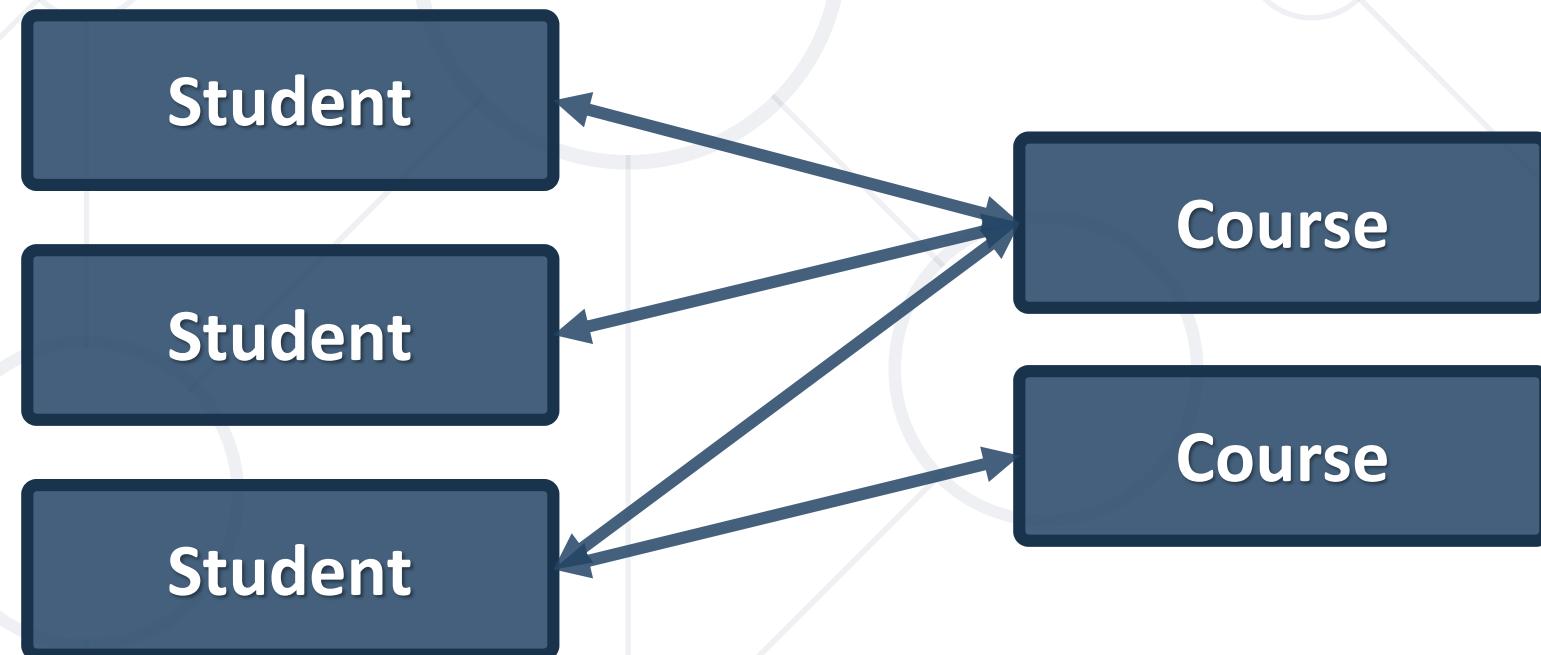
- **HasMany → WithOne / HasOne → WithMany**

```
modelBuilder.Entity<Department>()
    .HasMany(d => d.Employees)
    .WithOne(e => e.Department)
    .HasForeignKey(e => e.DepartmentId);
```

```
modelBuilder.Entity<Employee>()
    .HasOne(e => e.Department)
    .WithMany(d => d.Employees)
    .HasForeignKey(e => e.DepartmentId);
```

Many-to-Many

- Requires a **collection navigation property on both sides**
 - Implemented with collections in each entity, referring the other



Many-to-Many Implementation (1)

```
public class Course
{
    public string Name { get; set; }
    public ICollection<Student> Students { get; set; }
}
```

```
public class Student
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public ICollection<Course> Courses { get; set; }
}
```

Many-to-Many: Fluent API (1)

- Mapping **both sides** of relationship

```
builder.Entity<Student>()
    .HasMany(s => s.Courses)
    .WithMany(s => s.Students)
    .UsingEntity<Dictionary<string, object>>(
        "StudentCourse",
        r => r
            .HasOne<Course>()
            .WithMany()
            .HasForeignKey("CourseId")
            .HasConstraintName("FK_StudentCourse_Courses_CourseId")
            .OnDelete(DeleteBehavior.Cascade),
        r => r
            .HasOne<Student>()
            .WithMany()
            .HasForeignKey("StudentId")
            .HasConstraintName("FK_StudentCourse_Students_StudentId")
            .OnDelete(DeleteBehaviour.Cascade),
        j =>
        {
            j.HasKey("StudentId", "CourseId");
            j.ToTable("StudentsCourses");
            j.IndexerProperty<int>("StudentId").HasColumnName("StudentId");
            j.IndexerProperty<int>("CourseId").HasColumnName("CourseId");
        });
    
```

Composite Primary Key

Many-to-Many Implementation (2)

- You can optionally create a **join entity type**

```
public class Course
{
    public string Name { get; set; }
    public string Teacher { get; set; }

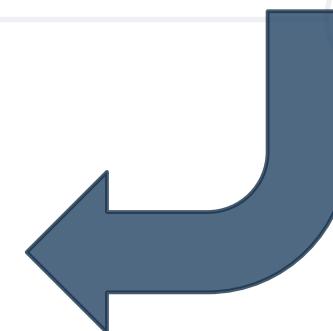
    public List<StudentCourse> StudentsCourses { get; set; }
}
```

```
public class Student
{
    public string FirstName { get; set; }
    public string LastName { get; set; }

    public List<StudentCourse> StudentsCourses { get; set; }
}
```

```
public class StudentCourse
{
    public int CourseId { get; set; }
    public Course Course { get; set; }

    public int StudentId { get; set; }
    public Student Student { get; set; }
}
```



Many-to-Many: Fluent API (2)

- Indirect many-to-many relationships

```
modelBuilder.Entity<StudentCourse>()
    .HasKey(sc => new { sc.StudentId, sc.CourseId });
```

Composite
Primary Key

```
builder.Entity<StudentCourse>()
    .HasOne(sc => sc.Student)
    .WithMany(s => s.StudentCourses)
    .HasForeignKey(sc => sc.StudentId);
```

```
builder.Entity<StudentCourse>()
    .HasOne(sc => sc.Course)
    .WithMany(s => s.StudentCourses)
    .HasForeignKey(sc => sc.CourseId);
```

Multiple Relations

- When two entities are related by more than one key
- Entity Framework needs help from **Inverse Properties**



Multiple Relations Implementation (1)

- **Person** Domain Model – defined as usual

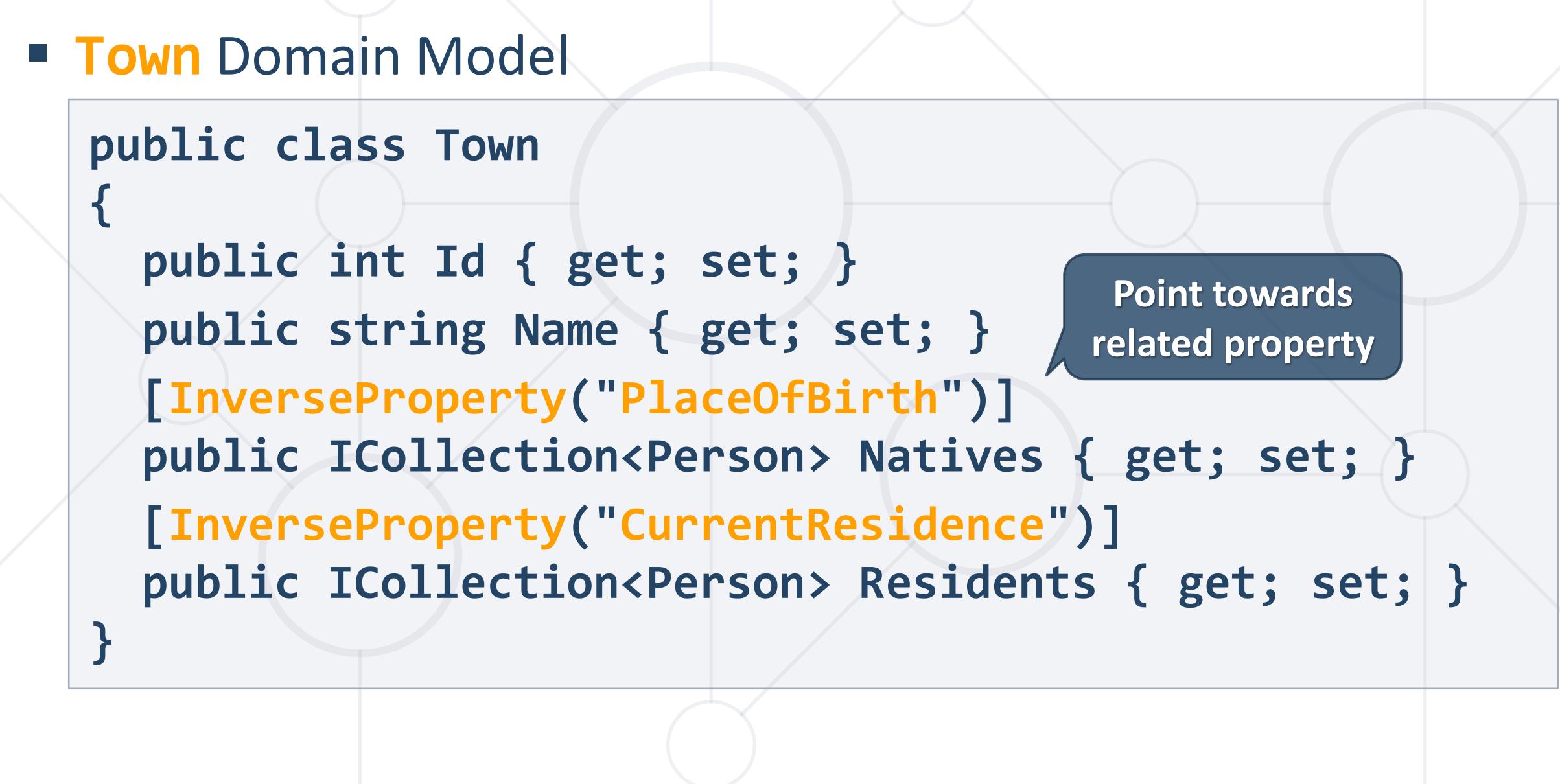
```
public class Person
{
    public int Id { get; set; }
    public string Name { get; set; }

    public Town PlaceOfBirth { get; set; }
    public Town CurrentResidence { get; set; }
}
```

Multiple Relations Implementation (2)

■ Town Domain Model

```
public class Town
{
    public int Id { get; set; }
    public string Name { get; set; }
    [InverseProperty("PlaceOfBirth")]
    public ICollection<Person> Natives { get; set; }
    [InverseProperty("CurrentResidence")]
    public ICollection<Person> Residents { get; set; }
}
```

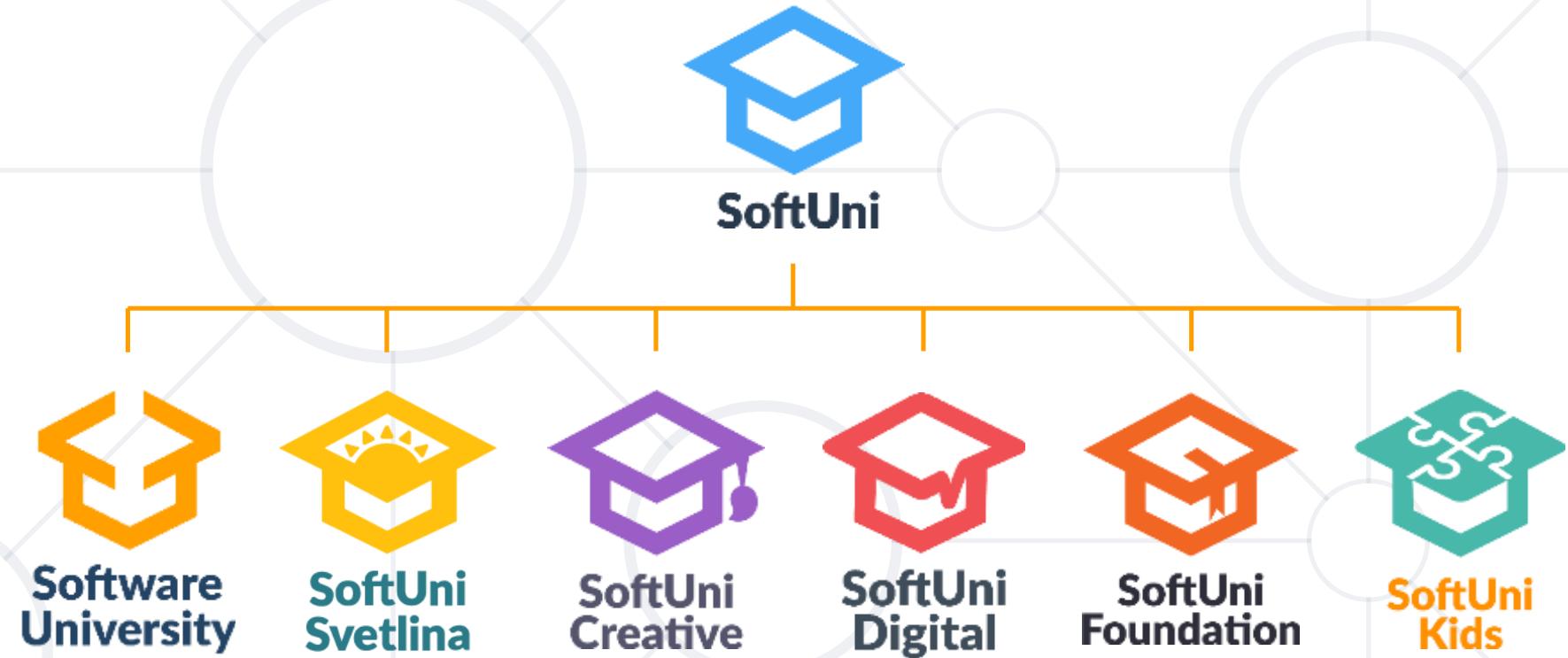


Point towards
related property

- The **Fluent API** gives us full control over Entity Framework object mappings
- **Attributes** can be used to express special table relationships and to customize entity behaviour
- Objects can be composed from other objects to represent complex **relationships**



Questions?



SoftUni Diamond Partners



SCHWARZ



Coca-Cola HBC
Bulgaria



Postbank

Решения за твоето упре



**SOFTWARE
GROUP**

Bosch.**.IO**



SmartIT



**PHAR
VISION**



CAREERS



INDEAVR
Serving the high achievers



**SUPER
HOSTING
.BG**



Educational Partners



Trainings @ Software University (SoftUni)



- 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
- Software University Forums
 - forum.softuni.bg



Software
University



- 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>



LINQ

Language Integrated Query in Entity Framework Core

SoftUni Team

Technical Trainers



SoftUni



Software University
<https://softuni.bg>

Table of Contents

- LINQ
 - Filtering
 - Select/Projection
 - Aggregation
 - Joining
 - SelectMany
- IEnumerable vs IQueryable
- Result Models

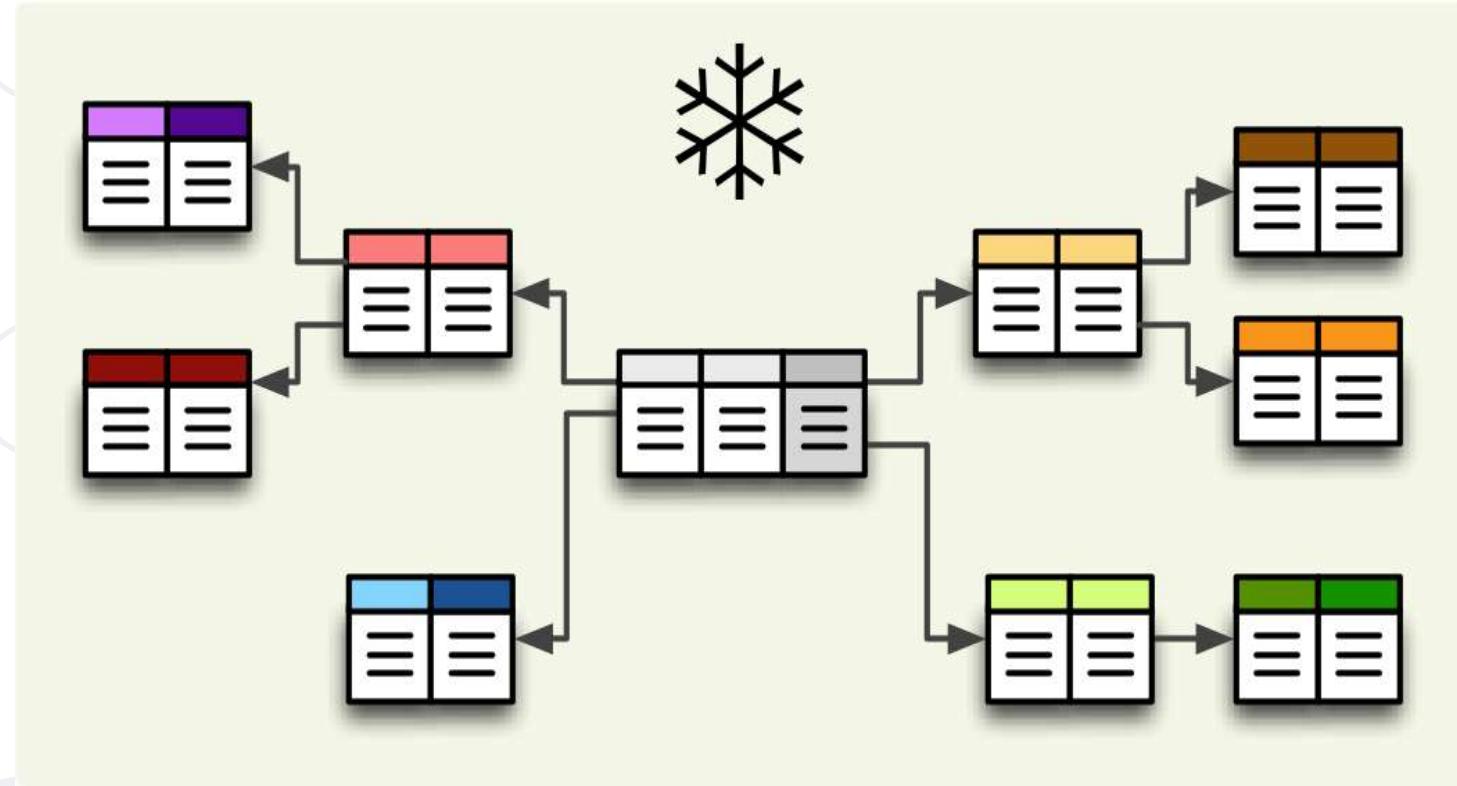


Have a Question?



sli.do

#csharp-db



Filtering and Aggregating Tables

Select, Join and Group Data Using LINQ

- **Where**

- Selects values that are based on a predicate function
- Syntax

```
string[] words = { "the", "quick", "brown", "fox", "jumps" };
```

```
IEnumerable<string> query =
    words.Where(word => word.Length == 3);
```

```
IEnumerable<string> query = from word in words
                             where word.Length == 3
                             select word;
```

Good Reasons to Use Select

- Limit network traffic by reducing the queried columns
- Syntax

```
var employeesWithTown = context
    .Employees
    .Select(employee => new
{
    EmployeeName = employee.FirstName,
    TownName = employee.Address.Town.Name
});
```

- SQL Server Profiler

```
SELECT [employee].[FirstName] AS [EmployeeName], [employee.Address.Town].[Name] AS [TownName]
FROM [Employees] AS [employee]
LEFT JOIN [Addresses] AS [employee.Address] ON [employee].[AddressID] = [employee.Address].[AddressID]
LEFT JOIN [Towns] AS [employee.Address.Town] ON [employee.Address].[TownID] =
[employee.Address.Town].[TownID]
```

Good Reasons Not to Use Select

- Data that is selected is **not** of the **initial entity type**

- **Anonymous type**, generated at runtime

```
[•] (local variable) System.Collections.Generic.List<'a> employeesWithTown
```

Anonymous Types:

```
'a is new { string EmployeeName, string TownName }
```

Local variable 'employeesWithTown' is never used

- **Data cannot be modified** (updated, deleted)
 - Entity is of a **different type**
 - Not associated with the **context** anymore

Aggregation

- Aggregate functions perform calculations on a set of input values and return a value
 - **Average** - Calculates the average value of a collection of values
 - **Count** - Counts the elements in a collection, optionally only those elements that satisfy a predicate function
 - **Max** and **Min** - Determine the maximum and the minimum value in a collection
 - **Sum** - Calculates the sum of the values in a collection

Joining Tables in EF: Using Join()

- Join tables in EF with **LINQ / extension methods** on **IEnumerable<T>** (like when joining collections)

```
var employees =  
    softUniEntities.Employees.Join(  
        softUniEntities.Departments,  
        (e => e.DepartmentID),  
        (d => d.DepartmentID),  
        (e, d) => new {  
            Employee = e.FirstName,  
            JobTitle = e.JobTitle,  
            Department = d.Name  
        }  
    );
```

Grouping Tables in EF

- Grouping also can be done by LINQ
 - The same way as with collections in LINQ
- Grouping with LINQ

```
var groupedEmployees =  
    from employee in softUniEntities.Employees  
    group employee by employee.JobTitle;
```

- Grouping with extension methods

```
var groupedCustomers = softUniEntities.Employees  
    .GroupBy(employee => employee.JobTitle);
```

SelectMany – Example (1)

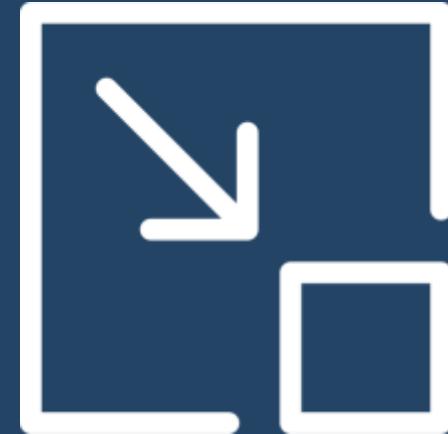
```
public class PhoneNumber
{
    public string Number { get; set; }
}
```

```
public class Person
{
    public IEnumerable<PhoneNumber> PhoneNumbers { get; set; }

    public string Name { get; set; }
}
```

SelectMany – Example (2)

```
IEnumerable<Person> people = new List<Person>();  
  
// "Select" gets a list of lists of phone numbers  
IEnumerable<IEnumerable<PhoneNumber>> phoneLists =  
    people.Select(p => p.PhoneNumbers);  
  
// "SelectMany" flattens it to just a list of phone numbers  
IEnumerable<PhoneNumber> phoneNumbers =  
    people.SelectMany(p => p.PhoneNumbers);  
  
// To include data from the parent in the result pass an expression  
// to the second parameter (resultSelector) in the overload  
var directory = people.SelectMany(p => p.PhoneNumbers,  
(parent, child) => new { parent.Name, child.Number });
```



IEnumerable vs IQueryable

- **IEnumerable<T>** is an interface that is available in the **System.Collection.Generic** namespace
- Implementation of the Iterator design pattern
- **IEnumerable** or **IEnumerable<T>** interface should be used only for **in-memory data objects**
- LINQ methods over **IEnumerable<T>** use **Func<>** parameters

- **IQueryable<T>** is an interface and it is available in **System.Linq**
- Provides functionality to evaluate queries against a specific **data source** where the type of the data may not be specified
- The **IQueryable** interface is intended for implementation by query providers
- LINQ methods over **IQueryable<T>** use **Expression<Func<>>** parameters (expression trees)
- Entity Framework can convert expression trees directly into SQL

- **`IEnumerable<T>`**
 - `System.Collections.Generic`
 - Base type for almost all .NET collections
 - LINQ methods works with `Func<>`
 - Good for **in-memory** data
- **`IQueryable<T>`**
 - `System.Linq` namespace
 - Derives the base interface from **`IEnumerable<T>`**
 - LINQ methods works with **`Expression<Func<>>`**
 - Good for queries over **data stores** such as databases



Result Models
Simplifying Models

Result Models (1)

- **Select()**, **GroupBy()** can work with **custom classes**
 - Allow you to **pass them** to methods and use them as a return type
 - Require some **extra code** (class definition)
- Sample Result Model



```
public class UserResultModel
{
    public string FullName { get; set; }
    public string Age { get; set; }
}
```

Result Models (2)

- Assign the fields as you would with an anonymous object

```
var currentUser = context.Users
    .Where(u => u.Id == 8)
    .Select(u => new UserResultModel
    {
        FullName = u.FirstName + " " + u.LastName,
        Age = u.Age
    })
    .SingleOrDefault();
```

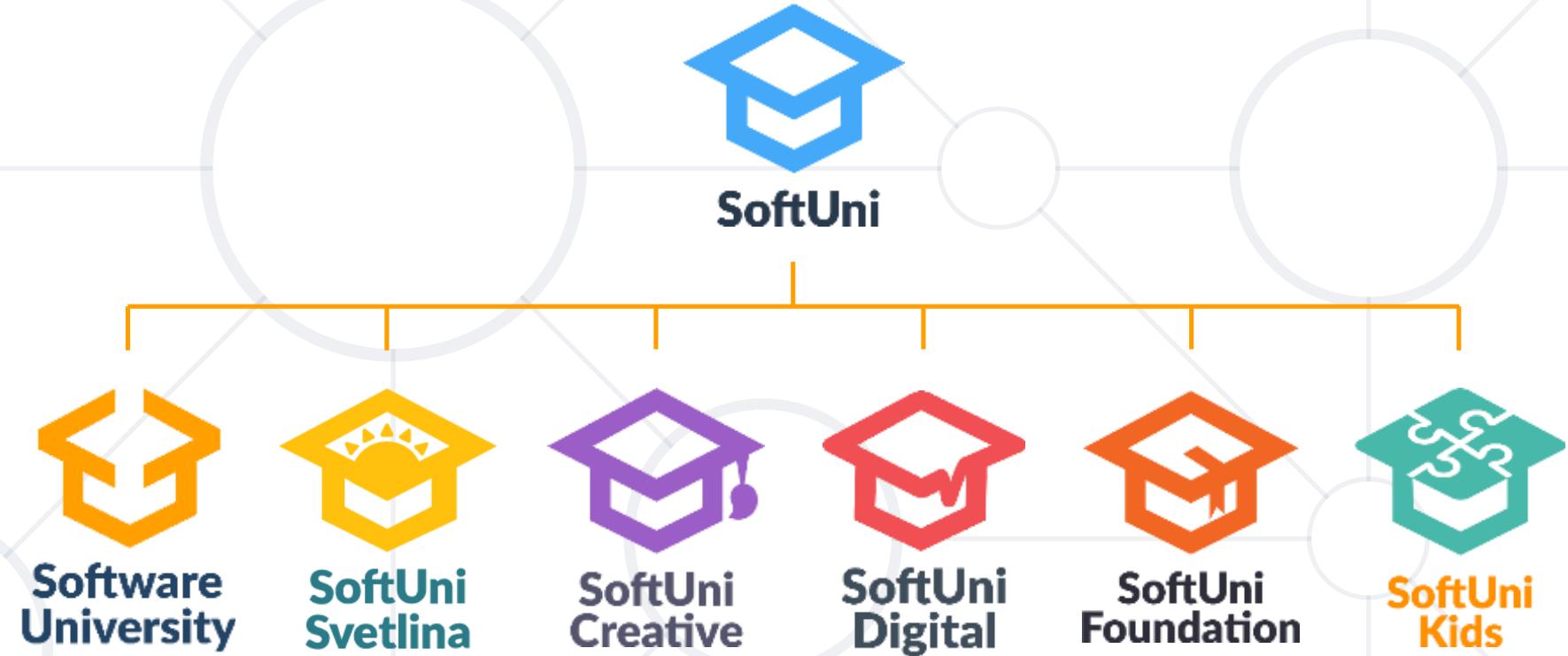
- The new type can be used in a method signature

```
public UserResultModel GetUserInfo(int Id) { ... }
```

- **LINQ**
 - Filtering, Aggregation, SelectMany, Joins
- **IEnumerable**
- **IQueryable**
- **Differences** between IEnumerable and IQueryable
- Result Models



Questions?



SoftUni Diamond Partners



SCHWARZ



Coca-Cola HBC
Bulgaria



Postbank

Решения за твоето упре



**SOFTWARE
GROUP**

Bosch.**.IO**



SmartIT



**PHAR
VISION**



CAREERS



INDEAVR
Serving the high achievers



**SUPER
HOSTING
.BG**

**DXC
TECHNOLOGY**

Educational Partners



- 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>



Trainings @ Software University (SoftUni)



- 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
- Software University Forums
 - forum.softuni.bg



Software
University



Advanced Querying

Advanced Entity Framework Core



SoftUni Team

Technical Trainers



SoftUni

Software University

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

Table of Contents

- Executing Native SQL Queries
 - Execute Stored Procedures
- Object State Tracking
- Bulk Operations
- Types of Loading
- Concurrency Checks
- Cascade Operations



Have a Question?



sli.do

#csharp-db



Executing Native SQL Queries

Parameterless and Parameterized

Executing Native SQL Queries

- Executing a **native SQL query** in EF Core directly

```
var query = "SELECT * FROM Employees";
var employees = db.Employees
    .FromSqlRaw(query)
    .ToArray();
```

- Limitations
 - **JOIN** statements **don't** get mapped to the entity class
 - **Required columns** must **always** be selected
 - **Target table** must be the same as the **DbSet**

Native SQL Queries with Parameters

- Native SQL queries can also be parameterized

```
var context = new SoftUniDbContext();
string nativeSQLQuery =
    "SELECT FirstName, LastName, JobTitle" +
    "FROM dbo.Employees WHERE JobTitle = {0}";
var employees = context.Employees.FromSqlRaw(
    nativeSQLQuery, "Marketing Specialist");
foreach (var employee in employees)
{
    Console.WriteLine(employee.FirstName);
}
```

Parameter
placeholder

Parameter
value

Interpolation in SQL Queries

- **FromSqlInterpolated** allows string interpolation syntax

```
var context = new SoftUniDbContext();
string jobTitle = "Marketing Specialist";
FormattableString nativeSQLQuery =
    $"SELECT * FROM dbo.Employees WHERE JobTitle = {jobTitle}";
var employees = context.Employees.FromSqlInterpolated(
    nativeSQLQuery);

foreach (var employee in employees)
{
    Console.WriteLine(employee.FirstName);
}
```



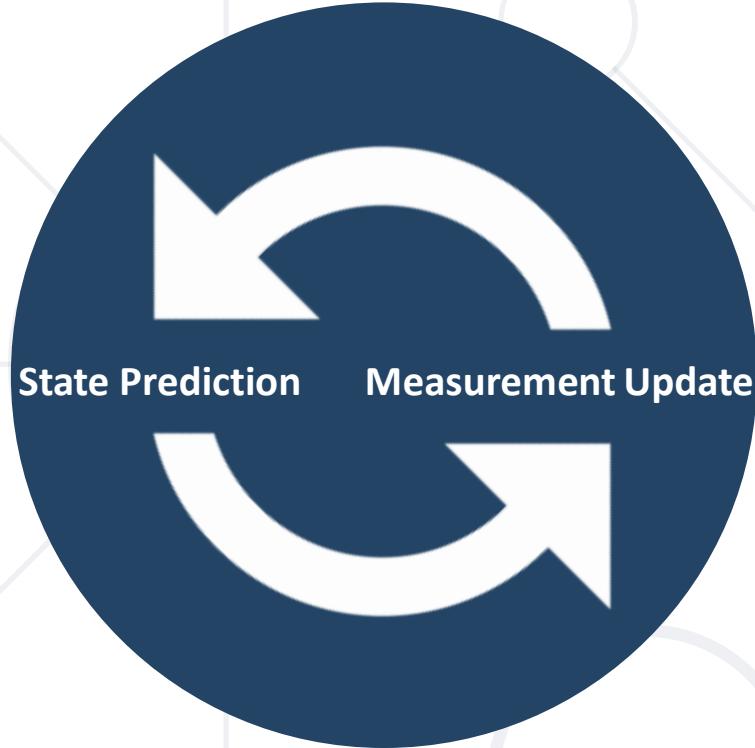
Interpolated parameter

Executing a Stored Procedure

- Stored Procedures can be executed via SQL

```
CREATE PROCEDURE UpdateSalary @param int  
AS  
UPDATE Employees SET Salary = Salary + @param;
```

```
var salaryParameter = new SqlParameter("@salary", 5);  
var query = "EXEC UpdateSalary @salary";  
context.Database.ExecuteSqlRaw(query, salaryParameter);
```



Object State Tracking

Attached and Detached Objects

- In Entity Framework, objects can be
 - **Attached** to the object context (tracked object)
 - **Detached** from an object context (untracked object)
- Attached objects are tracked and managed by the **DbContext**
 - **SaveChanges()** persists all changes in DB
- Detached objects are not referenced by the **DbContext**
 - Behave like a normal objects, which are not related to EF
 - We can get detached objects using **AsNoTracking()**
 - No-tracking queries are quicker to execute

Tracking and No-tracking Queries

- **Tracking** queries

```
var employee = context.Employees  
    .FirstOrDefault(e => e.EmployeeId == 1);  
  
employee.JobTitle = "Marketing Specialist";  
context.SaveChanges();
```

Returns attached entry

- **No-tracking** queries

```
var employees = context.Employees  
    .AsNoTracking()  
    .ToList();
```

Returns detached read-only entity

Attached Objects

- When a query is executed inside a **DbContext**, the returned objects are **automatically attached** to it
- When a context is **destroyed**, all **objects** in it are automatically **detached**
 - e.g., in **Web applications** between requests
- You might later **attach** objects that have been previously **detached** to a **new context**

Detaching Objects

- When is an object detached?
 - When we get the object from a **DbContext** and then **Dispose** it

```
Employee GetEmployeeById(int id)
{
    using (var SoftUniDbContext = new SoftUniDbContext())
    {
        return SoftUniDbContext.Employees
            .First(e => e.EmployeeID == id);
    }
}
```

Returned employee
is detached

- Manually: by setting its **State** to **Detached**

Reattaching Objects

- When we want to update a detached object, we need to **reattach it** and then update it: change to **Attached** state

```
void UpdateName(Employee employee, string newName)
{
    using (var softUniDbContext = new SoftUniDbContext())
    {
        var entry = softUniDbContext.Entry(employee);
        entry.State = EntityState.Modified;
        employee.FirstName = newName;
        softUniDbContext.SaveChanges();
    }
}
```



BULK

Bulk Operations

Multiple Update and Delete in Single Query

- EF Core **does not** support bulk operations
- **Z.EntityFramework.Plus** gives you the ability to perform **bulk update/delete** of entities

- Entity Framework Plus

```
Install-Package Z.EntityFramework.Plus.EFCore
```

- Read more: <https://entityframework-plus.net>

Bulk Delete

- Delete all users where **FirstName** matches given string

```
context.Employees  
    .Where(e => e.FirstName == "Pesho")  
    .Delete();
```



```
DELETE [dbo].[Employees]  
FROM [dbo].[Employees] AS j0 INNER JOIN (  
SELECT  
    [Extent1].[Id] AS [Id]  
    FROM [dbo].[Employees] AS [Extent1].[Name]  
    WHERE N'Pesho' = [Extent1].[Name]  
) AS j1 ON (j0.[Id] = j1.[Id])
```

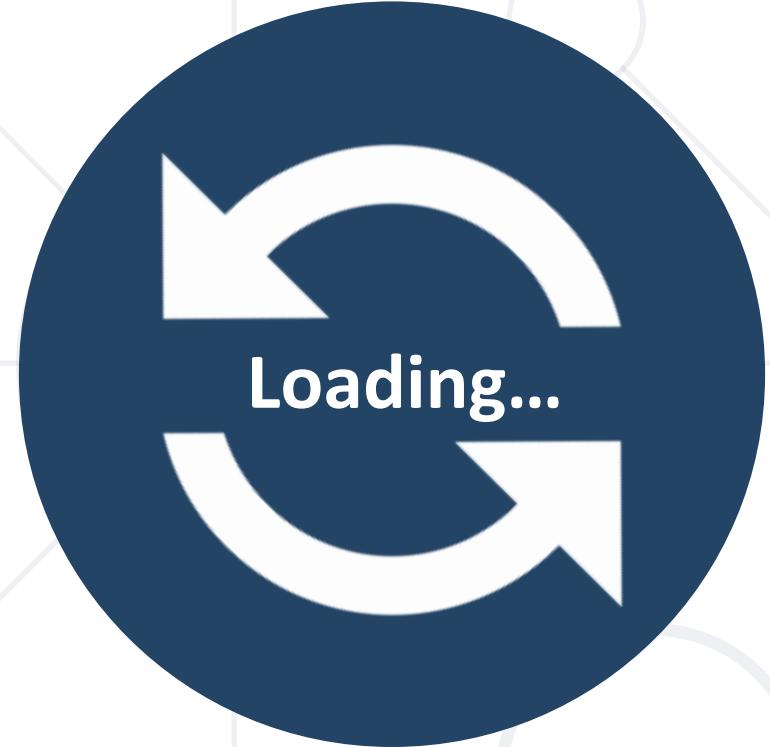
Bulk Update: Syntax

- Update all Employees with name "Niki" to "Stoyan"

```
context.Employees  
    .Where(t => t.Name == "Niki")  
    .Update(u => new Employee { Name = "Stoyan" });
```

- Update all Employees' age to 99 who have the name "Plamen"

```
IQueryable<Employee> employees = context.Employees  
    .Where(employee => employee.Name == "Plamen");  
  
employees.Update(employee => new Employee { Age = 99 });
```



Types of Loading

Lazy, Eager and Explicit Loading

Explicit Loading

- **Explicit loading** loads all records when they're needed
- Performed with the **.Reference().Load()** and **Collection().Load()** methods

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

```
context.Entry(employee)  
    .Reference(e => e.Department)  
    .Load();
```

```
context.Entry(employee)  
    .Collection(e => e.EmployeeProjects)  
    .Load();
```

Eager Loading

- Eager loading loads all related records of an entity at once
- Performed with the `Include()` and `ThenInclude()` methods

```
context.Towns.Include("Employees");
```

```
context.Towns.Include(town => town.Employees);
```

```
context.Employees  
    .Include(employee => employee.Address)  
    .ThenInclude(address => address.Town)
```

Lazy Loading

- Lazy Loading **delays** loading of data until it is used
- EF Core enables lazy-loading for any navigation property that can be **overridden (virtual)**
- Offers better performance in certain cases
 - Less RAM usage
 - Smaller result sets returned
- Each loading of navigational property is an addition query (N+1)

Enable Lazy Loading Proxies

- Install Lazy Loading Proxies

```
Install-Package Microsoft.EntityFrameworkCore.Proxies
```

- Enable the package

```
void OnConfiguring (DbContextOptionsBuilder options)
{
    options
        .UseLazyLoadingProxies()
        .UseSqlServer(myConnectionString);
}
```

N+1 Problem

- Refreshing the article list page, sends 11 queries to the database
 - The **first query** finds the first 10 articles
 - The subsequent **10 queries**, find each article's comments
 - Total of 11 queries ($N + 1$)





Concurrency Checks

Optimistic Concurrency Control in EF

- EF Core runs in **optimistic concurrency** mode (no locking)
 - By default, the conflict resolution strategy in EF is "**last one wins**"
 - The last change overwrites all previous concurrent changes
- Enabling "**first wins**" strategy for certain property in EF
 - **[ConcurrencyCheck]**

Last One Wins – Example

```
var contextFirst = new SoftUniDbContext();
var lastProjectFirstUser = contextFirst.Projects.First();
lastProjectFirstUser.Name = "Changed by the First User";

// The second user changes the same record
var contextSecondUser = new SoftUniDbContext();
var lastProjectSecond = contextSecondUser.Projects.First();
lastProjectSecond.Name = "Changed by the Second User";

// Conflicting changes: Last wins
contextFirst.SaveChanges();
contextSecondUser.SaveChanges();
```

Second user wins

First One Wins – Example

```
var context = new SoftUniDbContext();
var lastTownFirstUser = contextFirst.Towns.First();
lastTownFirstUser.Name = "First User";
```

[ConcurrencyCheck]

```
var contextSecondUser = new SoftUniDbContext();
var lastTownSecondUser = contextSecondUser.Towns.First();
lastTownSecondUser.Name = "Second User";
```

```
context.SaveChanges();
```

Changes get saved

```
contextSecondUser.SaveChanges();
```

DbUpdateConcurrencyException



cascade

Cascade Operations

Deleting Related Entities

Cascade Delete Scenarios

- Required FK with cascade delete set to true, deletes everything related to the deleted property
- Required FK with cascade delete set to false, throws exception (it cannot leave the navigational property with no value)
- Optional FK with cascade delete set to true, deletes everything related to the deleted property
- Optional FK with cascade delete set to false, sets the value of the FK to NULL

Cascade Delete with Fluent API (1)

- Using **OnDelete** with **DeleteBehavior** Enumeration:
 - **DeleteBehavior.Cascade**
 - Deletes related entities (default for required FK)
 - **DeleteBehavior.Restrict**
 - Throws exception on delete
 - **DeleteBehavior.ClientSetNull**
 - Default behavior for optional FK (does not affect database)
 - **DeleteBehavior.SetNull**
 - Sets the property to null (affects database)

Cascade Delete with Fluent API (2)

- Cascade delete syntax

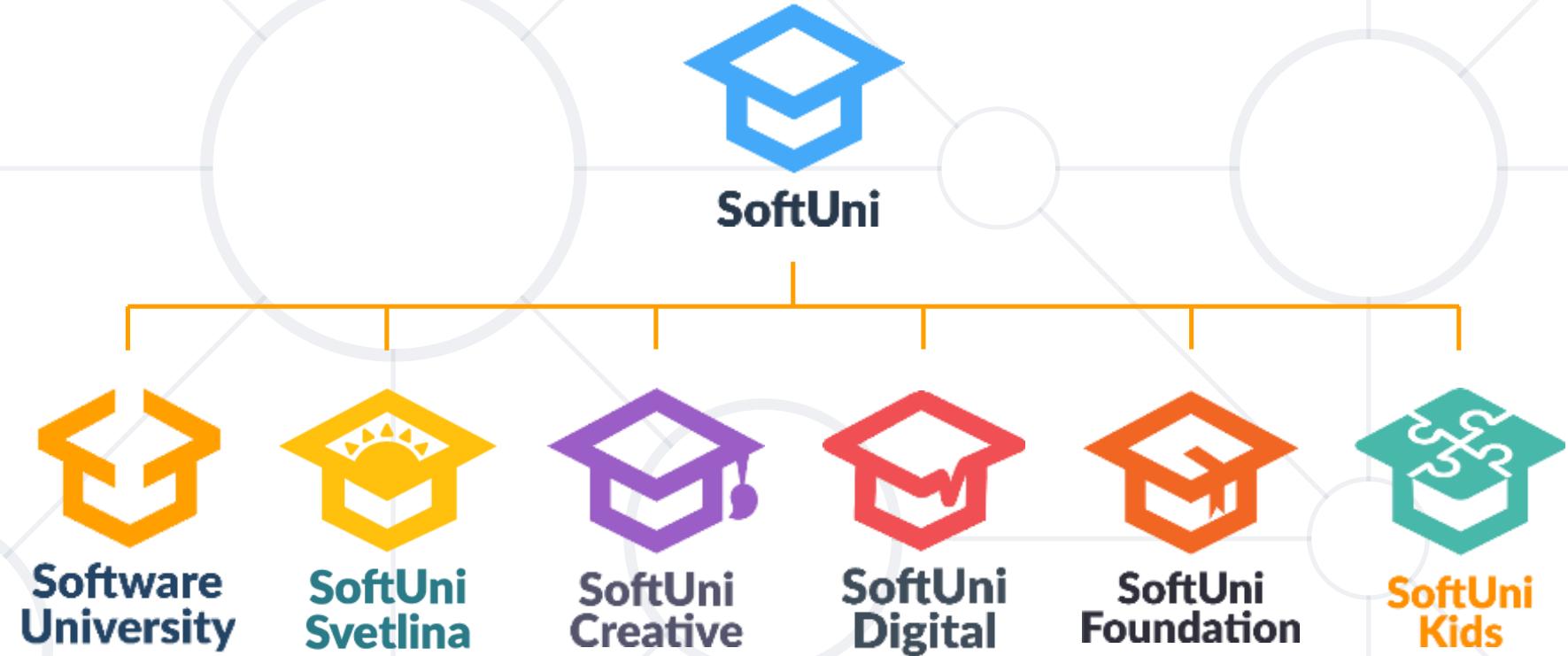
```
modelBuilder.Entity<User>()
    .HasMany(u => u.Replies)
    .WithOne(a => a.Author)
    .OnDelete(DeleteBehavior.Restrict);
```

```
modelBuilder.Entity<User>()
    .HasMany(u => u.Replies)
    .WithOne(a => a.Author)
    .OnDelete(DeleteBehavior.Cascade);
```

- Databases can be accessed directly with **SQL queries** from C# code
- EF keeps track of the **model state**
- **Entity Framework-Plus** lets you bundle **update** and **delete** operations
- EF supports lazy, eager and explicit **loading**
- With multiple users, **concurrency** of operations must be observed
- **Cascade delete** is on by default



Questions?



SoftUni Diamond Partners



SCHWARZ



Coca-Cola HBC
Bulgaria



Postbank

Решения за твоето упре



**SOFTWARE
GROUP**

Bosch.**.IO**



SmartIT

POKERSTARS

CAREERS

AMBITIONED

INDEAVR
Serving the high achievers

createX

**DRAFT
KINGS**

**SUPER
HOSTING
.BG**

Educational Partners



Trainings @ Software University (SoftUni)



- 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
- Software University Forums
 - forum.softuni.bg



Software
University



- 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>



C# Auto Mapping Objects

Manual Mapping
and AutoMapper Library



SoftUni Team

Technical Trainers



SoftUni



Software University

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

Table of Contents

- Data Transfer Objects
 - Manual Mapping
- AutoMapper Library
 - Mapping `ICollection<>`
 - Mapping `IQueryable<>`
 - Custom Member Mappings
 - Mapping Profiles



Have a Question?



sli.do

#csharp-db



Data Transfer Objects

Definition and Usage

What is a Data Transfer Object?

- A **DTO** is an object that **carries data** between processes
 - Used to **aggregate** only the **needed information** in a single call
 - Example: In web applications, between the **server** and **client**
- Doesn't contain any logic – only **stores values**

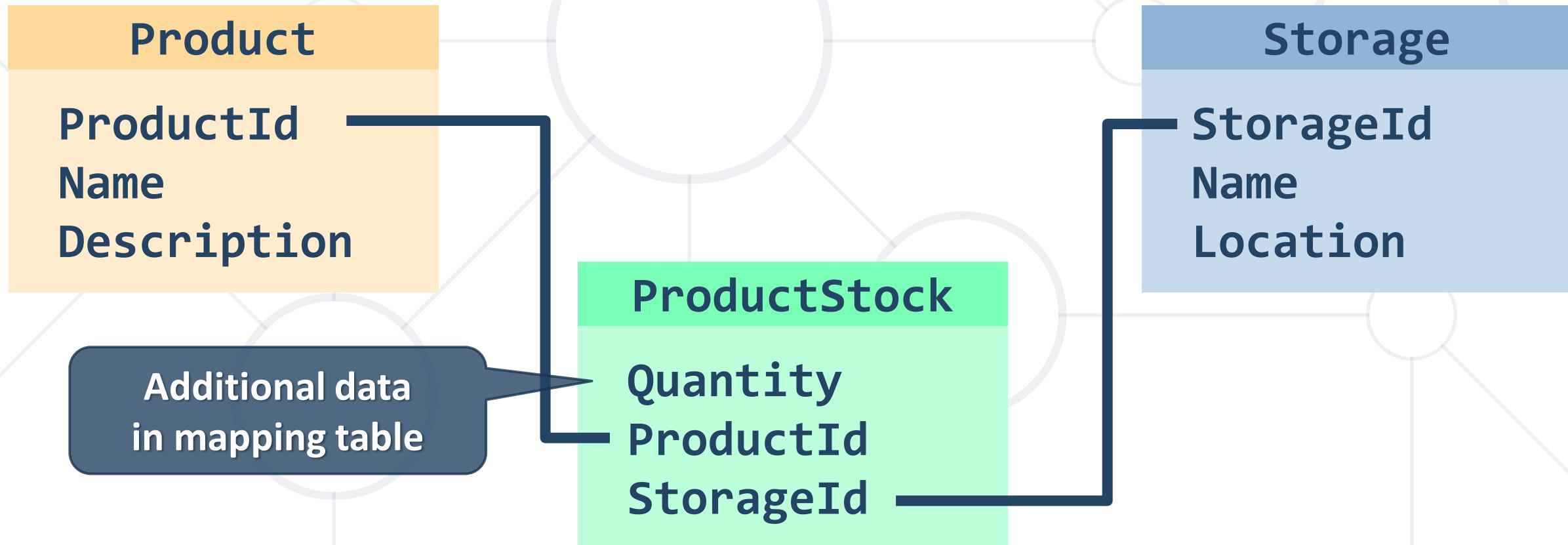
```
public class ProductDTO
{
    public string Name { get; set; }

    public int StockQty { get; set; }
}
```

- **Hide** particular properties that **clients** are not supposed to view
- **Remove** circular references
- **Omit** some properties in order to **reduce** payload **size**
- **Flatten** object graphs that contain nested objects to make them more convenient for clients (denormalization)
- **Decouple** your service layer from your database layer

Manual Mapping (1)

- Relationship Diagram



Manual Mapping (2)

- Get product name and stock quantity in a new DTO object

```
var product =  
    context.Products.FirstOrDefault();  
var productDto = new ProductDTO  
{  
    Name = product.Name,  
    StockQty = product.ProductStocks  
        .Sum(ps => ps.Quantity)  
};
```

Aggregate information from
mapping table



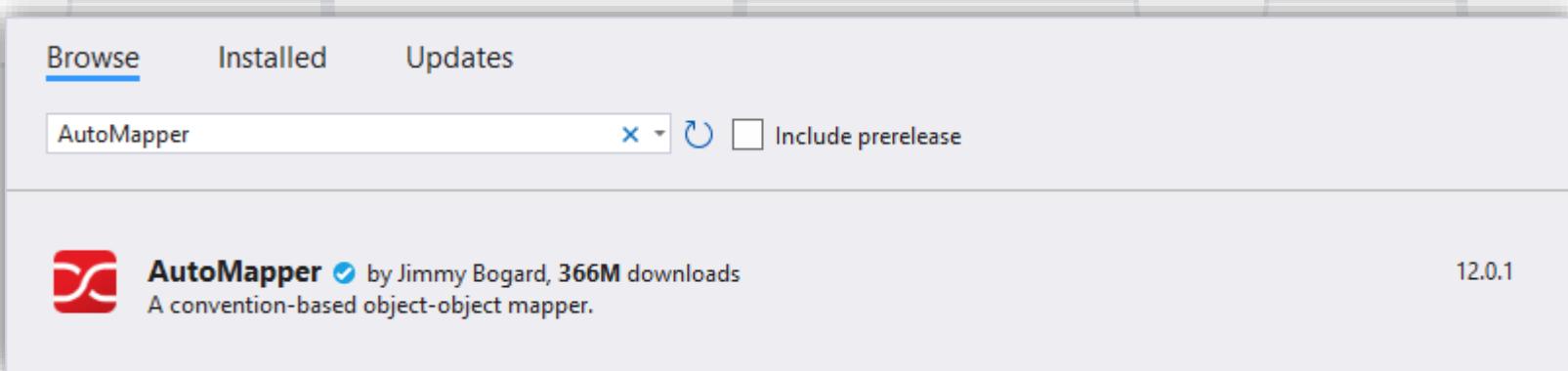
auto<red>x</red>mapper

AutoMapper Library

Automatic Translation of Domain Objects

What is AutoMapper?

- Library to eliminate **manual mapping** code
- Available as a **NuGet** Package



Install-Package AutoMapper

- Official [Website](#) and [GitHub](#)

Initialization and Configuration

- AutoMapper offers an **instance service** for use and configuration

- Add mappings between objects and DTOs

```
var config = new MapperConfiguration(cfg =>
    cfg.CreateMap<Product, ProductDTO>());
var mapper = config.CreateMapper();
```

Source

Target

- Properties will be mapped **by name**

```
var product = context.Products.FirstOrDefault();
ProductDTO dto = mapper.Map<ProductDTO>(product);
```

Multiple Mappings

- You can configure all mapping configurations at once

```
var config = new MapperConfiguration(cfg =>
{
    cfg.CreateMap<Product, ProductDTO>();
    cfg.CreateMap<Order, OrderDTO>();
    cfg.CreateMap<Client, ClientDTO>();
    cfg.CreateMap<SupportTicket, TicketDTO>();
});
var mapper = config.CreateMapper();
```

Mapping ICollection and IQueryable

- EF Core uses **IQueryable<T>** for all DB operations
 - AutoMapper can work with **IQueryable<T>** to map classes
- Using **AutoMapper** to map an entire DB collection

```
var posts = context.Posts
    .Where(p => p.Author.Username == "Nikolay.IT")
    .ProjectTo<PostDto>(config)
    .ToList();
```

IQueryable<PostDto>

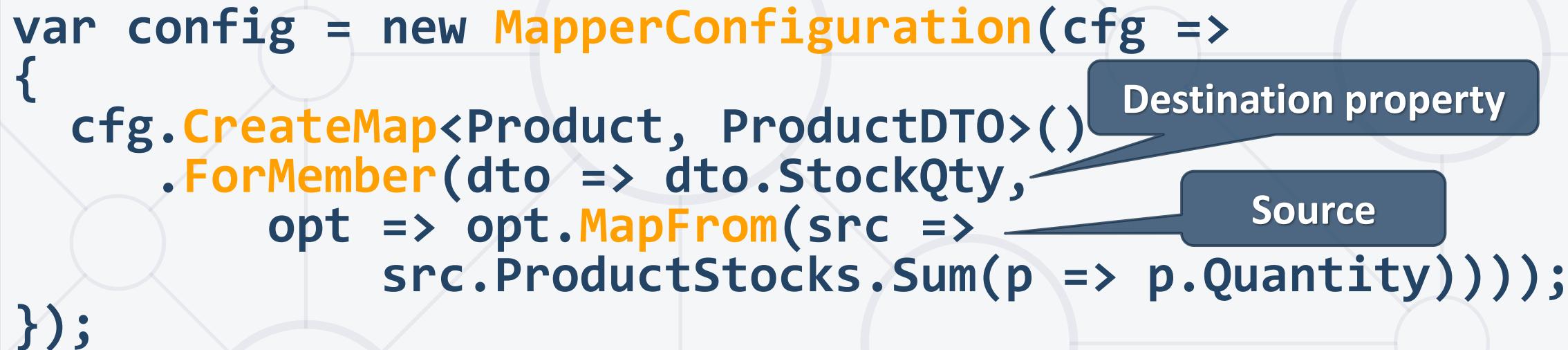
IQueryable<Post>

- Works like an automatic **.Select()**
 - AutoMapper helps EF to generate **optimized SELECT SQL query** (like projection with an **anonymous object**)

Custom Member Mapping

- Map properties that don't match naming convention

```
var config = new MapperConfiguration(cfg =>
{
    cfg.CreateMap<Product, ProductDTO>()
        .ForMember(dto => dto.StockQty,
            opt => opt.MapFrom(src =>
                src.ProductStocks.Sum(p => p.Quantity)));
});
```



Flattening Complex Objects

- **Flattening** of related objects is automatically supported

```
public class OrderDTO
{
    public string ClientName { get; set; }
    public decimal Total { get; set; }
}
```

- AutoMapper understands **ClientName** is the **Name** of a Client

```
var config = new MapperConfiguration(cfg =>
    cfg.CreateMap<Order, OrderDTO>());
var mapper = config.CreateMapper();
OrderDTO dto = mapper.Map<Order, OrderDTO>(order);
```

Unflattening Complex Objects

- **Unflattening** of related objects is automatically supported

```
public class OrderDTO
{
    public string ClientName { get; set; }
    public decimal Total { get; set; }
}
```

- AutoMapper understands **ClientName** is the **Name** of a **Client**, but to unflatten it, it needs **ReverseMap()**

```
var config = new MapperConfiguration(cfg =>
    cfg.CreateMap<Order, OrderDTO>().ReverseMap());
var mapper = config.CreateMapper();
Order order = mapper.Map<OrderDTO, Order>(dto);
```

Mapping Profiles

- We can extract our configuration to a class (called a **profile**)

```
public class ForumProfile : Profile
{
    public ForumProfile()
    {
        CreateMap<Post, PostDto>();
        CreateMap<Category, CategoryDto>();
    }
}
```

using AutoMapper;

- Using our configuration class

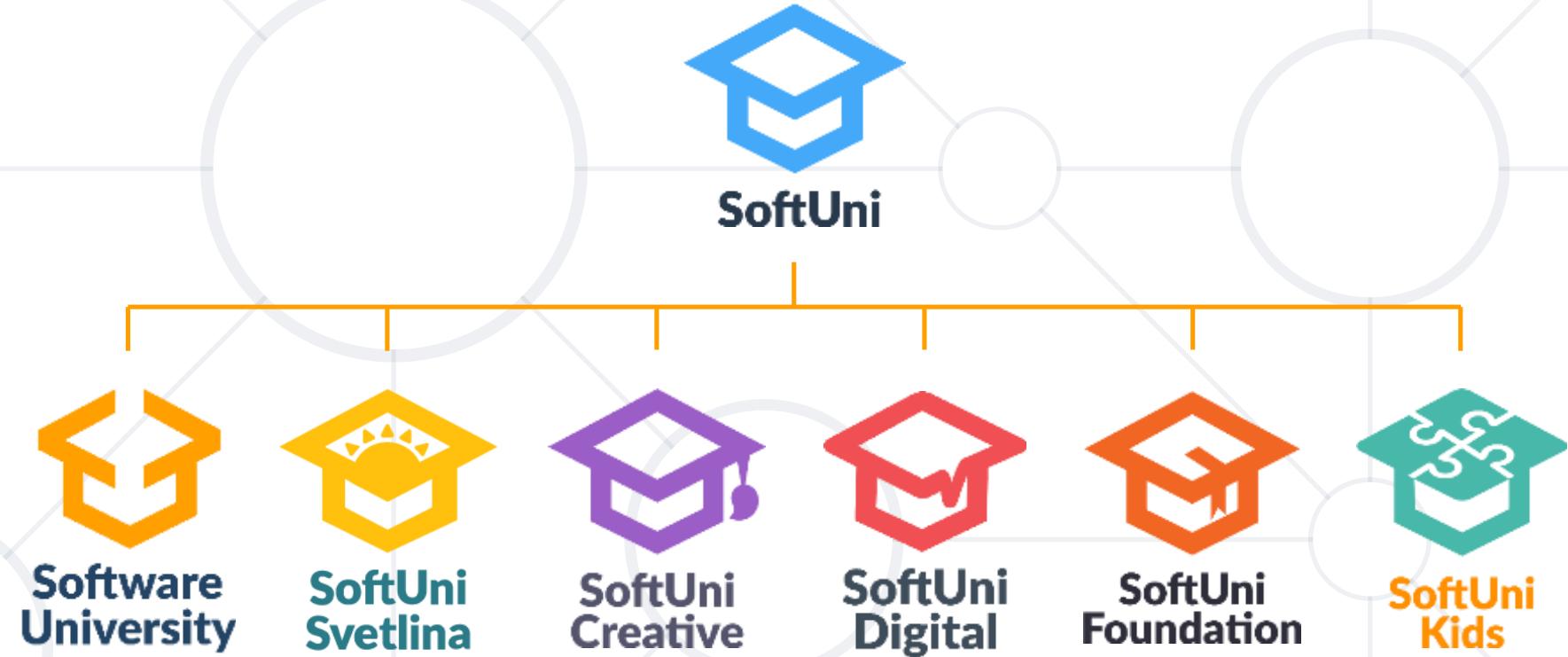
```
var config = new MapperConfiguration(cfg =>
    cfg.AddProfile<ForumProfile>());
```

Summary

- To reduce round-trip latency and payload size, data is transformed into a **DTO**
- **AutoMapper** is a library that automates this process and reduces boilerplate code
- Complex objects can be **flattened** to fractions of their sizes



Questions?



SoftUni Diamond Partners



SCHWARZ



Coca-Cola HBC
Bulgaria



Postbank

Решения за твоето упре



**SOFTWARE
GROUP**

Bosch.**.IO**



SmartIT



**PHAR
VISION**



CAREERS



INDEAVR
Serving the high achievers



**SUPER
HOSTING
.BG**



Educational Partners



- 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>



External Format Processing

Parsing JSON, JSON.NET



SoftUni



SoftUni Team

Technical Trainers



Software University

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

Table of Contents

- JSON Data Format
- Processing JSON
 - System.Text.Json
- JSON.NET
 - Configuring JSON.NET
 - LINQ-to-JSON
 - XML-to-JSON



Have a Question?



sli.do

#csharp-db



{JSON}

JSON Data Format

Definition and Syntax

JSON Data Format

- **JSON (JavaScript Object Notation)** is a lightweight data format
 - Human and machine-readable plain text
 - Based on **JavaScript** objects
 - Independent of development platforms and languages
 - JSON data consists of
 - Values (**strings, numbers, etc.**)
 - Key-value pairs: **{ key : value }**
 - Arrays: **[value1, value2, ...]**

```
{  
    "firstName": "Pesho",  
    "courses": ["C#", "JS", "ASP.NET"],  
    "age": 23,  
    "hasDriverLicense": true,  
    "date": "2012-04-23T18:25:43.511Z",  
    // ...  
}
```

JSON Data Format (2)

- The JSON data format follows the rules of object creation in JS

- Strings, numbers and Booleans** are valid JSON

```
"this is a string and is valid JSON"
```

```
3.14
```

```
true
```

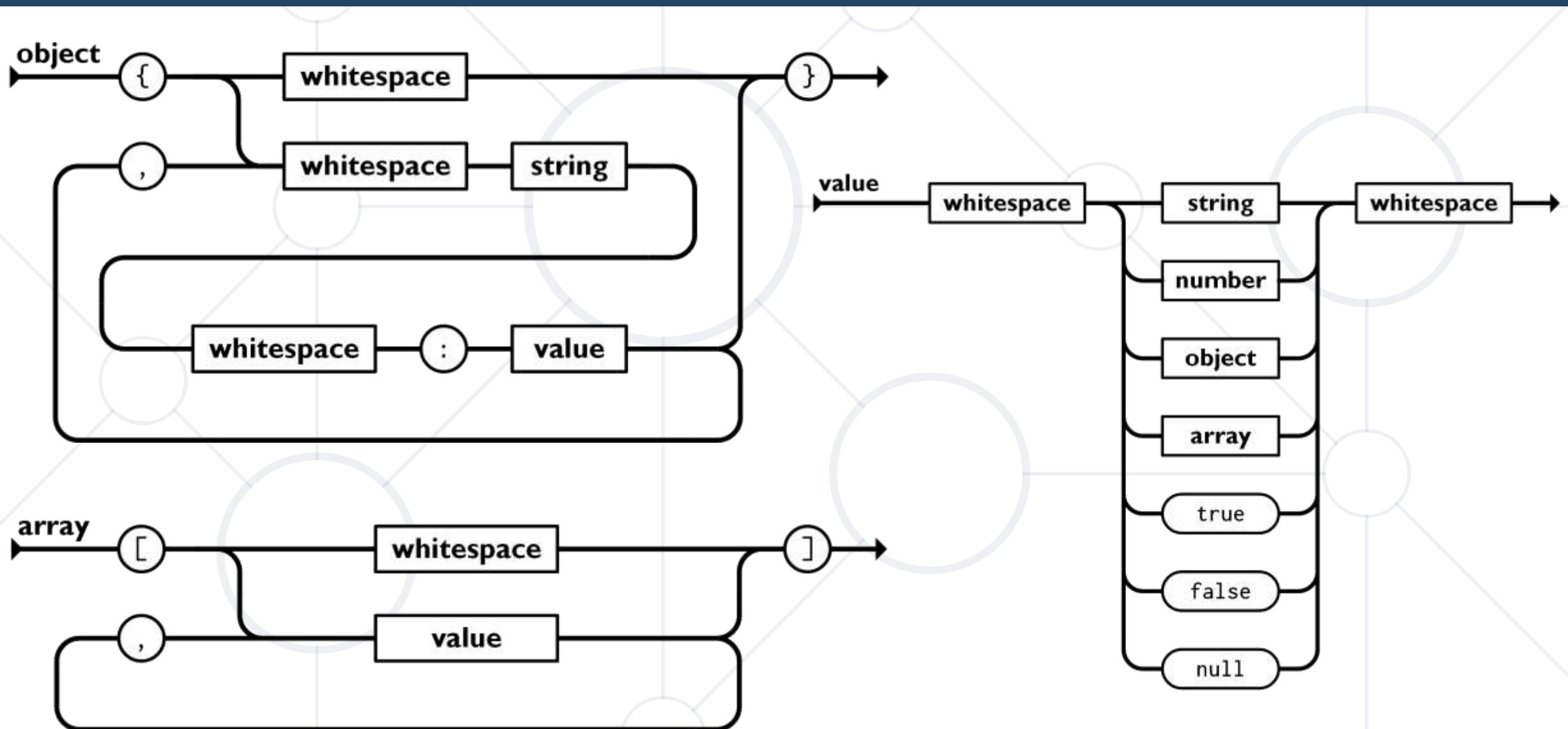
- Arrays** are valid JSON

```
[5, "text", true]
```

- Objects** are valid JSON (key-value pairs)

```
{
  "firstName": "Svetlin", "lastName": "Nakov",
  "jobTitle": "Technical Trainer", "age": 40
}
```

Object, Array and Value in JSON



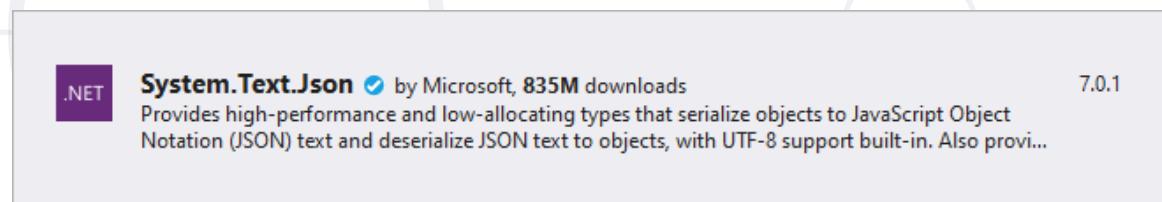


Processing JSON

Parsing JSON in C# and .NET with `System.Text.Json`

Built-in JSON Support

- .NET has built-in JSON support through the
System.Text.Json NuGet Package



- It supports **serializing** objects and **deserializing** (parsing) strings
- Include **the following namespaces** into your project

```
using System.Text.Json;
```

```
using System.Text.Json.Serialization;
```



Serializing JSON

- The **System.Text.Json** serializer can read and write JSON

```
class WeatherForecast
{
    public DateTime Date { get; set; } = DateTime.Now;
    public int TemperatureC { get; set; } = 30;
    public string Summary { get; set; } = "Hot summer day";
}

static void Main()
{
    WeatherForecast forecast = new WeatherForecast();
    string weatherInfo = JsonSerializer.Serialize(forecast);
    Console.WriteLine(weatherInfo);
}
```

Serializing JSON (2)

- Creating a JSON file

```
static void Main()
{
    WeatherForecast forecast = new WeatherForecast();
    string weatherInfo = JsonSerializer.Serialize(forecast);
    File.WriteAllText(file, weatherInfo);
}
```



```
{"Date": "2020-07-16T13:33:25", "TemperatureC": 30, "Summary": "Hot summer day"}
```

Deserializing JSON

- To deserialize from a file, we read the file into a string and then use the **Deserialize** method

```
static void Main()
{
    string jsonString = File.ReadAllText(file);
    WeatherForecast forecast =
        JsonSerializer.Deserialize<WeatherForecast>(jsonString);
}
```



Name	Value	Type
forecast	{JsonDemo.Program.WeatherForecast}	JsonDemo.Program.WeatherForecast
Date	{5.2.2023 r. 20:16:32}	System.DateTime
Summary	"Hot summer day"	string
TemperatureC	30	int



JSON.NET

Better JSON Parsing for .NET Developers

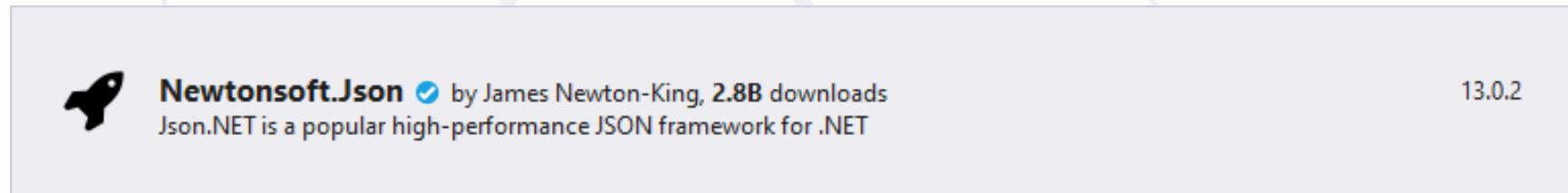
What is JSON.NET?

- JSON.NET is a JSON **framework** for .NET
 - More functionality than built-in functionality
 - Supports **LINQ-to-JSON**
 - Out-of-the-box support for parsing between **JSON** and **XML**
 - Open-source project: <http://www.newtonsoft.com>
 - **Newtonsoft.Json** vs **System.Text.Json**
 - [Performance comparison](#)
 - [Table of differences](#)



Installing JSON.NET

- To install JSON.NET use the **NuGet Package Manager**



- Or with a command in the Package Manager Console

```
Install-Package Newtonsoft.Json
```

General Usage

- JSON.NET exposes a static service **JsonConvert**
- Used for parsing and configuration to
 - **Serialize** an object

```
var jsonProduct = JsonConvert.SerializeObject(product);
```

- **Deserialize** an object

```
var objProduct =  
    JsonConvert.DeserializeObject<Product>(jsonProduct);
```

JSON.NET Features

- JSON.NET can be configured to
 - **Indent** the output JSON string
 - Convert JSON to **anonymous types**
 - Control the **casing** and **properties** to parse
 - Skip errors
- JSON.NET also supports
 - LINQ-to-JSON
 - Direct parsing between XML and JSON

Configuring JSON.NET (1)

- By default, the result is a **single line of text**
- To indent the output string use **Formatting.Indented**

```
JsonConvert.SerializeObject(products, Formatting.Indented);
```

```
{
  "pump": {
    "Id": 0,
    "Name": "Oil Pump",
    "Description": null,
    "Cost": 25.0
  },
  "filter": {
    "Id": 0,
    "Name": "Oil Filter",
    "Description": null,
    "Cost": 15.0
  }
}
```

Configuring JSON.NET (2)

- Deserializing to **anonymous types**



```
var json = @"{ 'firstName': 'Svetlin',
    'lastName': 'Nakov',
    'jobTitle': 'Technical Trainer' }";
```

```
var template = new
{
    FirstName = string.Empty,
    LastName = string.Empty,
    JobTitle = string.Empty
};
```

```
var person = JsonConvert.DeserializeObjectAnonymousType(json,
template);
```

Incoming JSON

Template objects

JSON.NET Attributes

- By default JSON.NET takes each property / field from the class and parses it
 - This can be controlled using **attributes**

```
public class User
{
    [JsonProperty("user")]
    public string Username { get; set; }

    [JsonIgnore]   Skip the property
    public string Password { get; set; }
}
```

Parse Username
to user

Skip the property

JSON.NET Parsing of Objects

- By default JSON.NET takes each property / field from the class and parses it
 - This can be controlled using **ContractResolver**

```
DefaultContractResolver contractResolver =
    new DefaultContractResolver()
{
    NamingStrategy = new SnakeCaseNamingStrategy()
};
var serialized = JsonConvert.SerializeObject(person,
    new JsonSerializerSettings()
{
    ContractResolver = contractResolver,
    Formatting = Formatting.Indented
});
```

LINQ-to-JSON (1)

- LINQ-to-JSON works with **JObjects**

- Create from JSON string

```
JObject obj = JObject.Parse(jsonProduct);
```

- Reading from file

```
var people = JObject.Parse(File.ReadAllText(@"c:\people.json"))
```

- Using **JObject**

```
foreach (JToken person in people)
{
    Console.WriteLine(person["FirstName"]); // Ivan
    Console.WriteLine(person["LastName"]); // Petrov
}
```

- **JObjects** can be queried with LINQ

```
var json = JObject.Parse(@"{'products': [
    {'name': 'Fruits', 'products': ['apple', 'banana']},
    {'name': 'Vegetables', 'products': ['cucumber']}]}");

var products = json["products"].Select(t =>
    string.Format("{0} ({1})",
        t["name"],
        string.Join(", ", c["products"]))
);

// Fruits (apple, banana)
// Vegetables (cucumber)
```

XML-to-JSON

```
string xml = @"<?xml version='1.0' standalone='no'?>
<root>
    <person id='1'>
        <name>Alan</name>
        <url>www.google.com</url>
    </person>
    <person id='2'>
        <name>Louis</name>
        <url>www.yahoo.com</url>
    </person>
</root>";

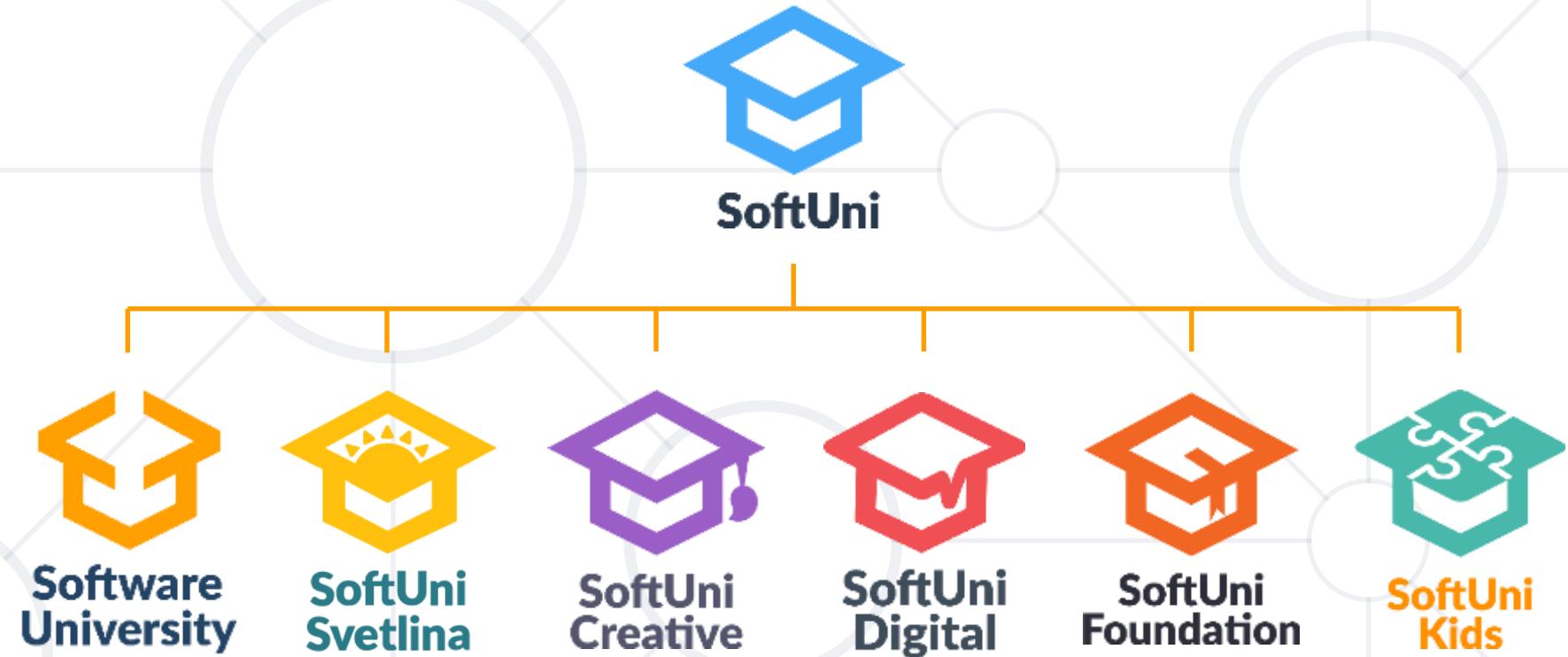
 XmlDocument doc = new XmlDocument();
 doc.LoadXml(xml);
 string jsonText = JsonConvert.SerializeXmlNode(doc);
```

```
{
    "?xml": {
        "@version": "1.0",
        "@standalone": "no"
    },
    "root": {
        "person": [
            {
                "@id": "1",
                "name": "Alan",
                "url": "www.google.com"
            },
            {
                "@id": "2",
                "name": "Louis",
                "url": "www.yahoo.com"
            }
        ]
    }
}
```

- **JSON** is a cross platform text-based data format
- **System.Text.Json** is the JSON Parser in C#
- **JSON.NET** is a fast framework for working with JSON data



Questions?



SoftUni Diamond Partners



SCHWARZ



Coca-Cola HBC
Bulgaria



Postbank

Решения за твоето упре



Bosch..IO****

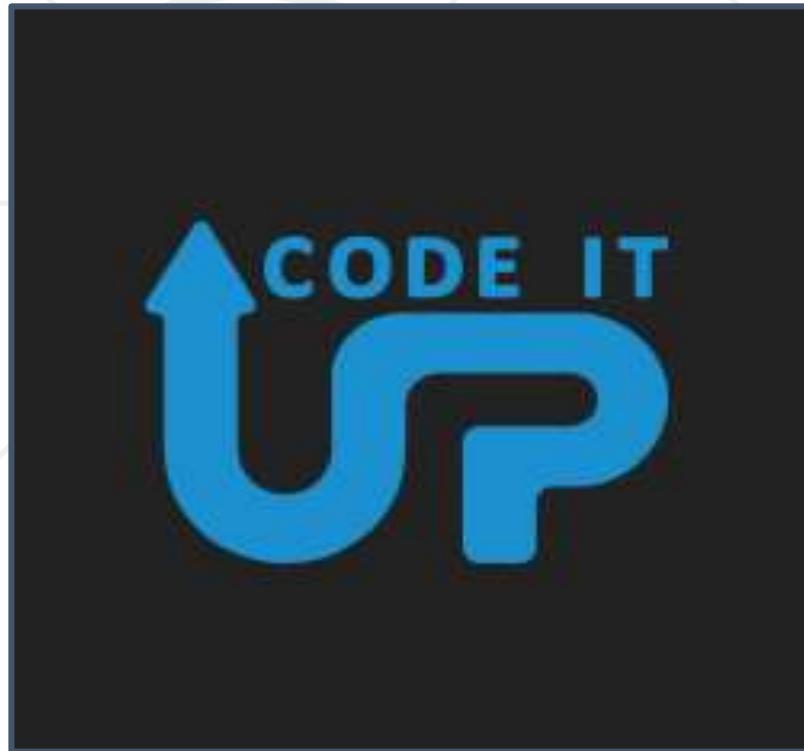


INDEAVR
Serving the high achievers



**SUPER
HOSTING
.BG**

Educational Partners



Trainings @ Software University (SoftUni)



- 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
- Software University Forums
 - forum.softuni.bg



Software
University



- 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>



XML Processing

Parsing XML

XDocument and LINQ-to-XML



SoftUni

SoftUni Team

Technical Trainers

 Software
University



Software University

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

Table of Contents

- What is XML?
- Parsing XML
- XML Serialization
- XML Deserialization
- XML Attributes



Have a Question?



sli.do

#csharp-db



What is XML?

Format Description and Application

What is XML?

- EXtensible Markup Language
 - Universal notation (data format / language) for describing structured data using text with tags
 - Designed to store and transport data
 - The data is stored together with the meta-data about it



XML - Example

```
<?xml version="1.0"?>
<library name="Developer's Library">
  <book>
    <title>Professional C# and .NET</title>
    <author>Christian Nagel</author>
    <isbn>978-0-470-50225-9</isbn>
  </book>
  <book>
    <title>Teach Yourself XML in 10
Minutes</title>
    <author>Andrew H. Watt</author>
    <isbn>978-0-672-32471-0</isbn>
  </book>
</library>
```

Root
(document)
element

XML header
tag (prolog)

Attribute
(key / value pair)

Element

Opening tag

Element value

Closing tag

XML Syntax

- **Header** – defines a **version** and character **encoding**

```
<?xml version="1.0" encoding="UTF-8"?>
```

- **Elements** – define the structure
- **Attributes** – element metadata
- **Values** – actual data, that can also be nested elements

Element name

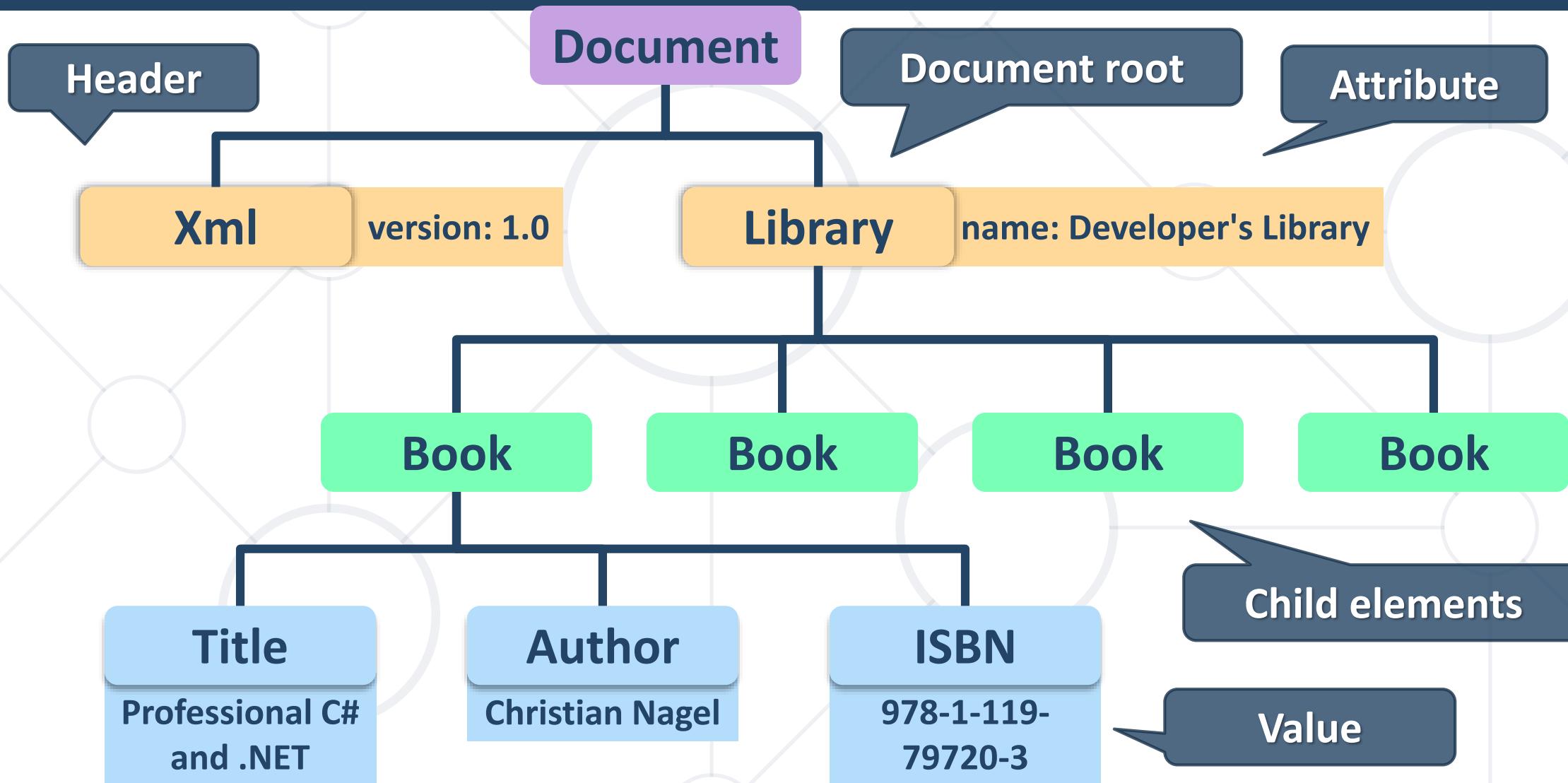
Attribute

Value

```
<title lang="en">Professional C# and .NET</title>
```

- **Root element** – required to **only** have **one**

XML – Structure



■ Similarities

- Both are **text based** notations
- Both use **tags** and **attributes**

■ Differences

- HTML describes documents, XML is a syntax for describing other languages (**meta-language**)
- HTML describes the **layout** and the structure of information
- XML requires the documents to be **well-formatted**



XML: Advantages

- XML is **human-readable** (unlike binary formats)
- Stores any kind of **structured data**
- Data comes with self-describing **meta-data**
- Full Unicode support
- Custom XML-based languages can be designed for certain apps
- **Parsers** available for virtually all languages and platforms



XML: Disadvantages

- XML data is **bigger** (takes more space) than binary or JSON
 - More memory consumption, more network traffic, more hard-disk space, more resources, etc.
- **Decreased performance**
 - CPU consumption: need of parsing / constructing the XML tags
- XML is **not** suitable for **all** kinds of **data**
 - E.g., binary data: graphics, images, videos, etc.

XML vs JSON

■ XML

- XML data is typeless
- All XML data should be string
- Data needs to be parsed
- Supports comments
- Supports various encoding

■ JSON

- JSON object has a type
- JSON types: string, number, array, Boolean
- Data is accessible as JSON objects
- Doesn't support comments
- Supports only UTF-8 encoding





Parsing XML

Using XDocument and LINQ

LINQ-to-XML

- LINQ-to-XML
 - Use the power of **LINQ** to process XML data
 - Easily read, search, write, modify XML documents
- LINQ-to-XML classes
 - **XDocument** – represents a LINQ-enabled XML document (containing prolog, root element, ...)
 - **XElement** – main component holding information
 - **XAttribute** – XML attributes information



- To process an XML string

```
string str = @"<?xml version=""1.0""?>
<!-- comment at the root level --&gt;
&lt;Root&gt;
    &lt;Child&gt;Content&lt;/Child&gt;
&lt;/Root&gt;";
XDocument doc = XDocument.Parse(str);</pre>
```

- Loading XML directly from file

```
XDocument xmlDoc = XDocument.Load("../..../books.xml");
```

Working with XDocument (1)

```
var cars = xmlDoc.Root.Elements();  
foreach (var car in cars)  
{  
    string make = car.Element("make").Value;  
    string model = car.Element("model").Value;  
    Console.WriteLine($"{make} {model}");  
}
```

Access root element

Get collection of children

Access element by name

Get value

Working with XDocument (2)

- Set an element value by name
 - If it doesn't exist, it will be **added**
 - If it is set to **null**, it will be **removed**

```
customer.SetElementValue("birth-date", "1990-10-04T00:00:00");
```

- Remove an element from its parent

```
var youngDriver = customer.Element("is-young-driver");
youngDriver.Remove();
```

Working with XDocument (3)

- Get or set an element attribute by name

```
customer.Attribute("name").Value
```

- Get a list of all attributes for an element

```
var attrs = customer.Attributes();
```

- Set an attribute value by name

- If it doesn't exist, it will be **added**

- If it is set to **null**, it will be **removed**

```
customer.SetAttributeValue("age", "21");
```

LINQ-to-XML - Searching with LINQ

- Searching in XML with LINQ is like searching with LINQ in array

```
XDocument xmlDoc = XDocument.Load("cars.xml");
var cars = xmlDoc.Root.Elements()
    .Where(e => e.Element("make").Value == "Opel" &&
        long.Parse(e.Element("travelled-distance").Value) >= 30000)
    .Select(c => new
    {
        Model = c.Element("model").Value,
        Traveled = c.Element("travelled-distance").Value
    })
    .ToList();
foreach (var car in cars)
    Console.WriteLine(car.Model + " " + car.Traveled);
```

Creating XML with XElement

- **XDocuments** can be composed from **XElements** and **XAttributes**

```
<books>
  <book>
    <author>Don Box</author>
    <title lang="en">ASP.NET</title>
  </book>
</book>
```

```
XDocument xmlDoc = new XDocument();
xmlDoc.Add(
  new XElement("books",
    new XElement("book",
      new XElement("author", "Don Box"),
      new XElement("title", "ASP.NET", new XAttribute("lang", "en")))
  ));
```

Add as root

Added with value

Optional attribute

Serializing XML to File

- To flush an **XDocument** to file with default settings

```
xmlDoc.Save("myBooks.xml");
```

- To disable automatic indentation

```
xmlDoc.Save("myBooks.xml", SaveOptions.DisableFormatting);
```

- To serialize **any object** to file

```
var serializer = new XmlSerializer(typeof(ProductDTO));
using (var writer = new StreamWriter("myProduct.xml"))
{
    serializer.Serialize(writer, product);
}
```

Deserialize XML from String XML

- To **deserialize** an object from an XML string

```
var serializer = new XmlSerializer(typeof(OrderDto[]),  
    new XmlRootAttribute("Orders"));  
  
var deserializedOrders =  
(OrderDto[])serializer.Deserialize(new StringReader(xmlString));
```

- Specifying **root attribute** name

```
var attr = new XmlRootAttribute("Orders");  
var serializer = new XmlSerializer(typeof(OrderDto[]), attr);  
  
var deserializedOrders =  
(OrderDto[])serializer.Deserialize(new StringReader(xmlString));
```



XML Attributes

Using Xml Attributes

XML Attributes

- We can use several attributes to control serialization to XML
 - `[XmlAttribute("Name")]` – Specifies the type's **name** in XML
 - `[XmlElement("name")]` – Serializes as **XML Attribute**
 - `[XmlElement]` – Serialize as **XML Element**
 - `[XmlAttribute]` – **Do not** serialize
 - `[XmlElement]` – Serialize as an **array** of XML elements
 - `[XmlAttribute]` – Specifies the **root** element name
 - `[XmlAttribute]` – Serialize **multiple xml elements** on **one line**

XML Attributes: Example

- We can use several XML attributes to control serialization

```
[XmlAttribute("Book")]
public class BookDto
{
    [XmlAttribute("name")]
    public string Name { get; }

    [XmlElement("Author")]
    public string Author { get; }

    [XmlAttribute("Price")]
    public decimal Price { get; }
}
```

XML Type name

Not serialized



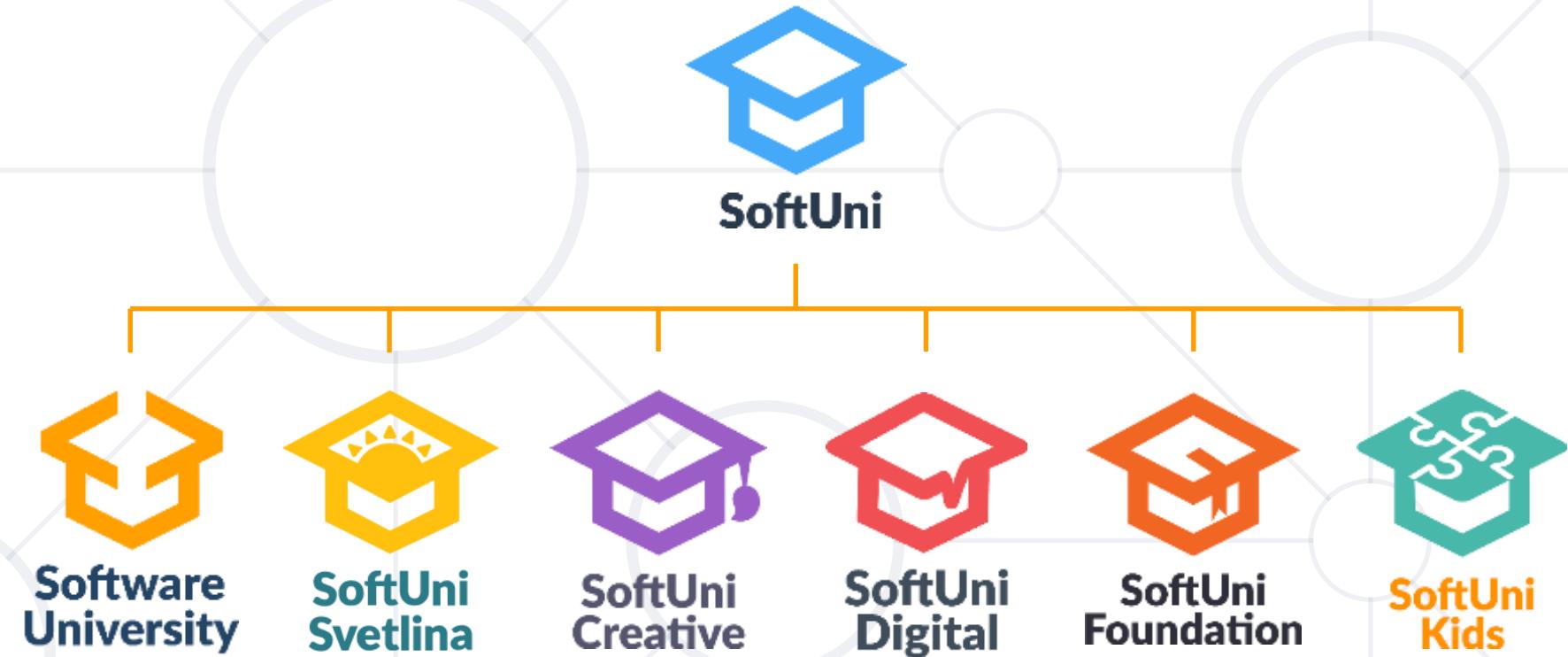
```
<Book name="It">
    <Author>Stephen King</Author>
</Book>
<Book name="Frankenstein">
    <Author>Mary Shelley</Author>
</Book>
<Book name="Queen Lucia">
    <Author>E.F. Benson</Author>
</Book>
<Book name="Paper Towns">
    <Author>John Green</Author>
</Book>
```

Summary

- **XDocument** is a system object for working with XML in .NET, which supports LINQ
- XML can be read and saved **directly to file**
- **XML** can be serialized to and from **class**
- **XML Attributes** are easy way to describe the **XML file**



Questions?



SoftUni Diamond Partners



SCHWARZ



Coca-Cola HBC
Bulgaria



Postbank

Решения за твоето упре



Bosch..IO****



INDEAVR
Serving the high achievers



**SUPER
HOSTING
.BG**

Educational Partners



Trainings @ Software University (SoftUni)



- 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
- Software University Forums
 - forum.softuni.bg



Software
University



- 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>



Best Practices and Architecture

Useful Patterns and Code Structure



SoftUni



Software University

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

SoftUni Team

Technical Trainers



Table of Contents

- Project Structure
- EF Core Optimizations
- Useful Patterns

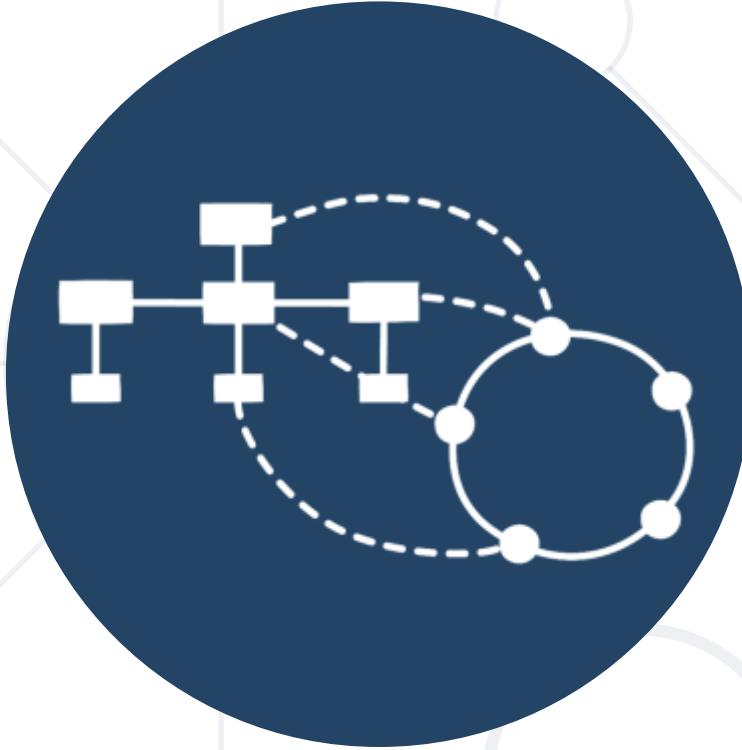


Have a Question?



sli.do

#CSharpDB

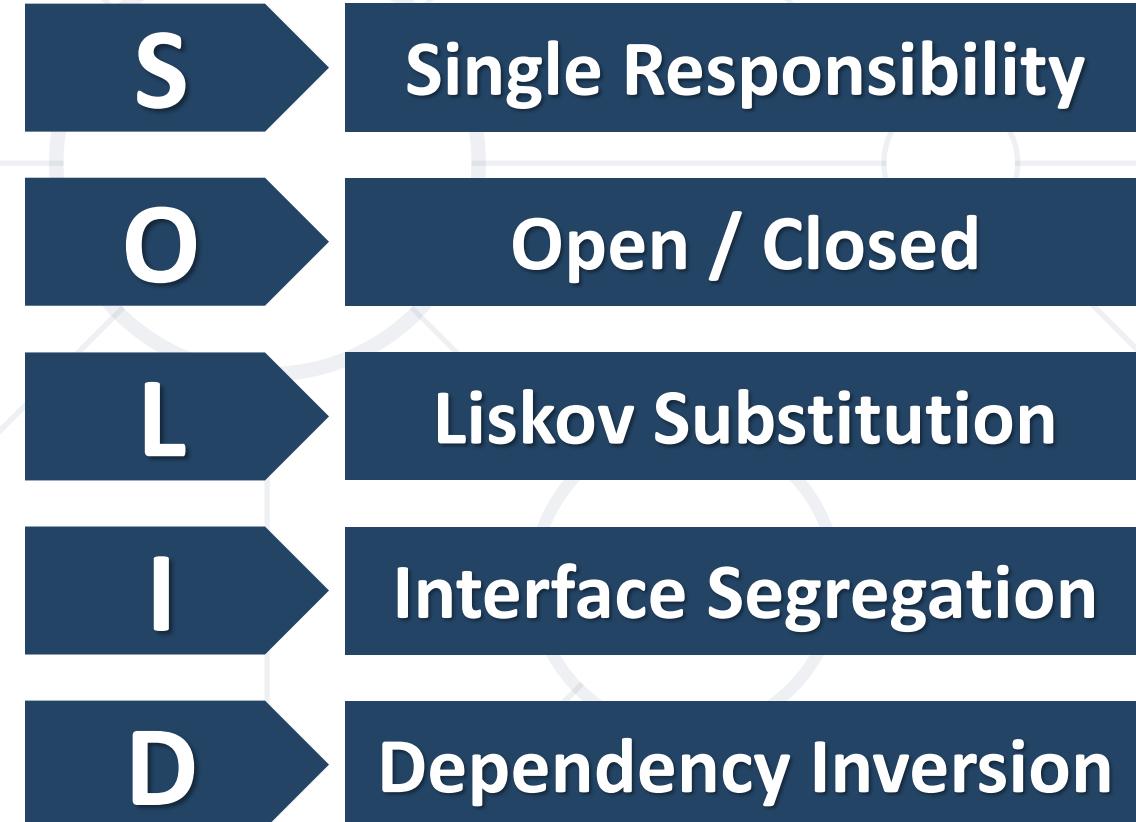


Project Structure

Organizing Solutions

Importance of Organized Code

- Scalability
- Maintainability
- Manageability
- Testability



Project Organization

- Application code can be split into sections
 - **Data Layer** – database connection (context)
 - **Domain Models** – entity classes
 - **Client** – user-interaction and app logic
 - **Business Logic** – data validation, transformations
- Reasons
 - Easier to locate files when maintaining
 - Don't have to rebuild entire codebase after changes (DLLs)



Usage Optimization

Entity Framework Core Performance

Usage Optimization (1)

- Only fetch **required data** by filtering and projecting your queries

```
context.Employees  
    .Where(e => e.Salary >= 15000)  
    .Select(e => new {  
        e.FirstName,  
        e.LastName,  
        e.Salary  
    })  
);
```

SELECT

```
1 AS [C1],  
[Extent1].[FirstName] AS [FirstName],  
[Extent1].[LastName] AS [LastName],  
[Extent1].[Salary] AS [Salary]  
FROM [dbo].[Employees] AS [Extent1]  
WHERE [Extent1].[Salary] >= cast(15000 as decimal(18))
```

Usage Optimization (2)

- LINQ queries are **executed** each time the data is **accessed**
 - If materialized in a collection – **ToList()**
 - If the elements are aggregated – **Count()**, **Average()**,
First()
 - When a property is accessed
- Try to delay execution (materialization) until you actually need the results
- You can monitor query execution using **Express Profiler**

Usage Optimization (3)

- EF will cache entities and compare the cache for changes
 - Use **Find()** with change detection disabled

```
try
{
    context.ChangeTracker.AutoDetectChangesEnabled = false;
    var product = context.Products.Find(productId);
    ...
}
finally
{
    context.ChangeTracker.AutoDetectChangesEnabled = true;
}
```

Usage Optimization (4)

- When adding or updating a record, Entity Framework makes a call to **DetectChanges()**
- Use **AddRange()** and **RemoveRange()** to reduce calls

```
List<Product> products = new  
List<Product>()  
{ product1, product2, product3 };  
  
context.Products.AddRange(products);
```

Works with
any collection

Usage Optimization (5)

- Entity Framework builds **associations** and **tracks changes** for every loaded entity
- If we **only** want to **display** data, this process is redundant
- Disable tracking

```
context.Products  
    .AsNoTracking()  
    .Where(p => p.Price < 150)  
    .ToList();
```

- Note this also **disables caching**!

Loading Methods (1)

- Payload size and number of roundtrips to the database are inversely proportional
 - **Lazy** – less data, more queries
 - **Eager** – more data, less queries
- There is no best approach – performance depends on usage scenario

Loading Methods (2)

- Do you need to access many **navigation properties** from the fetched entities?
 - No – **Lazy** for large payloads, **Eager** for small
 - Yes – **Eager** loading for up three entities, **Lazy** for more
- Do you know exactly what data will be needed at run time?
 - No – **Lazy**
 - Yes – **Eager** at first unless, **Lazy** if loading lots of data

Loading Methods (3)

- Is your code executing far from your database? (increased network latency)
 - **No** – **Lazy** will simplify your code; don't take database proximity for granted
 - **Yes** – Depending on scenario **Eager** will require fewer round trips
 - Always test application-wide performance, only optimize if results aren't satisfactory



Design Patterns

Solving Problems More Easily

Design Patterns

- **Singleton** – Ensure a class has only one instance and provide a global point of access to it
- **Service Locator** – Make a service available globally and decouple the calling class from the dependent object
- **Dependency Injection** - no client code has to be changed simply because an object it depends on needs to be changed to a different one
- **Command** – Encapsulate a request as an object, allowing delayed execution, undo and replay
- **Repository** – Separates the data access logic and maps it to the entities in the business logic
- **Unit of work** – Used to group one or more into a single transaction or "unit of work", so that all operations either pass or fail as one

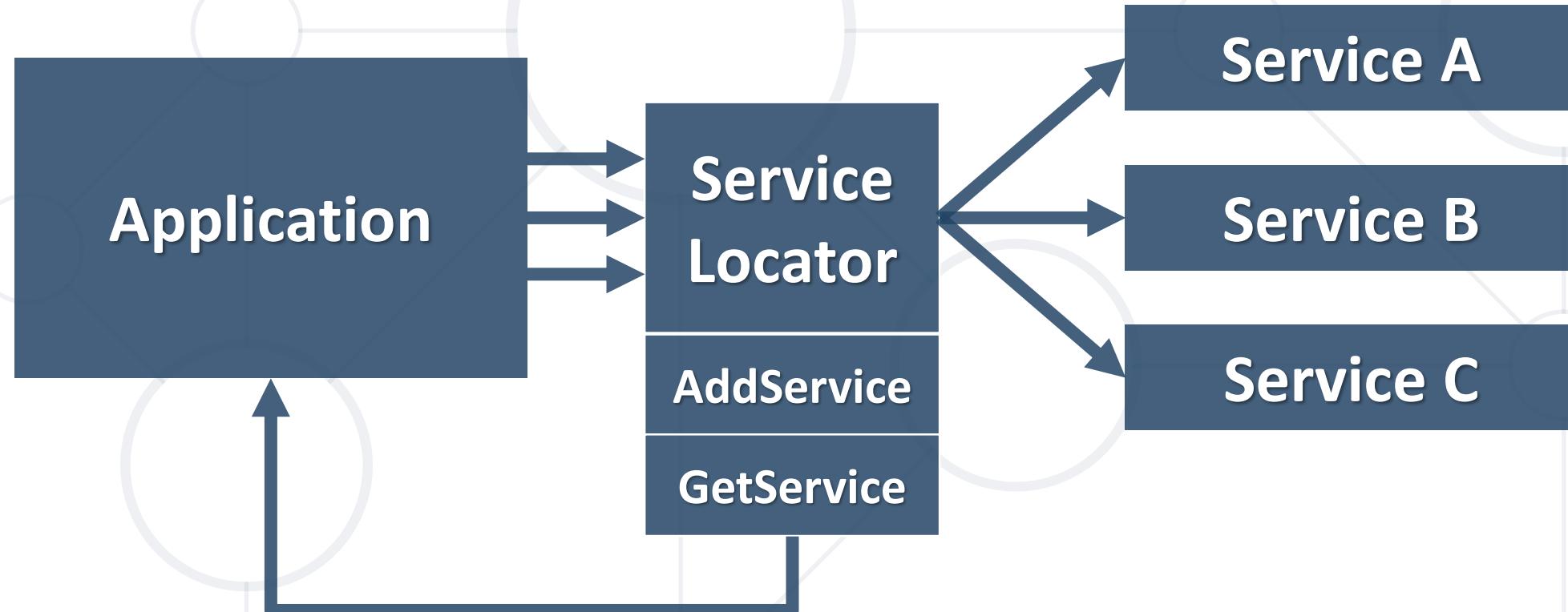
Singleton Pattern

```
public class Authenticator
{
    private static Authenticator instance;
    private Authenticator() { ... }
    public static Authenticator Instance
    {
        get
        {
            if (instance == null)
                instance = new Authenticator();
            return instance;
        }
    }
}
```

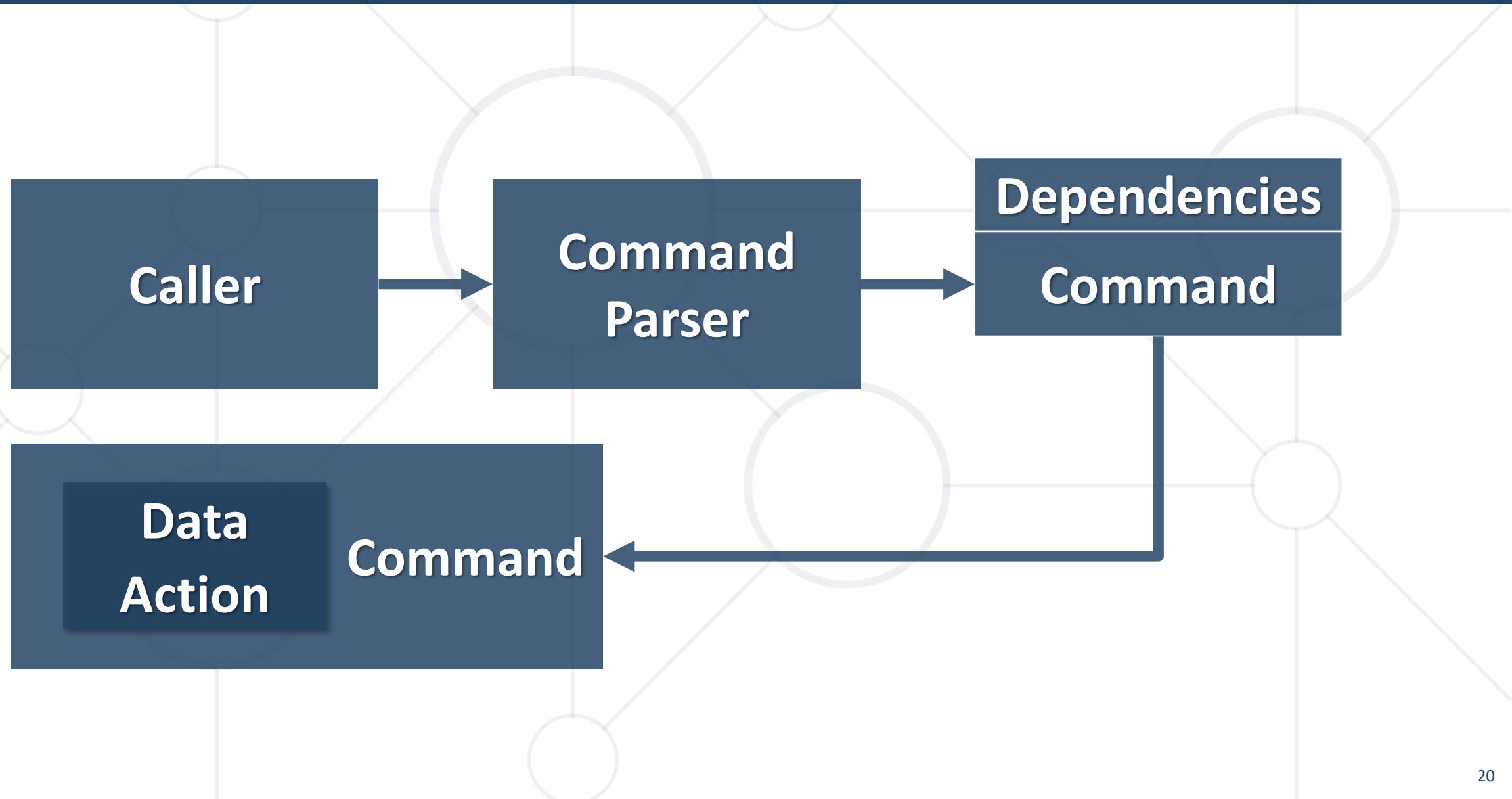
Private Constructor

Instantiate when first
accessed

Service Locator



Command Pattern

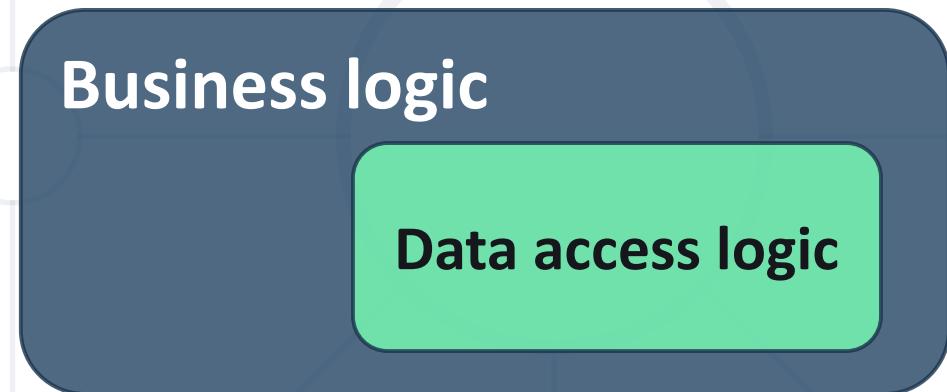


Repository Pattern

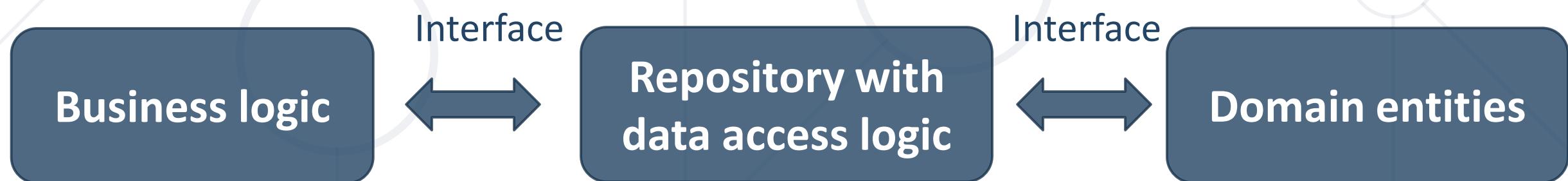
- It works with the **domain entities** and performs **data access logic**
- The domain entities, the data access logic and the business logic talk to each other **using interfaces**
- It **hides the details** of data access from the business logic
- Business logic **can access** the data object without having knowledge of the underlying data access architecture

Repository Pattern

- Without repository

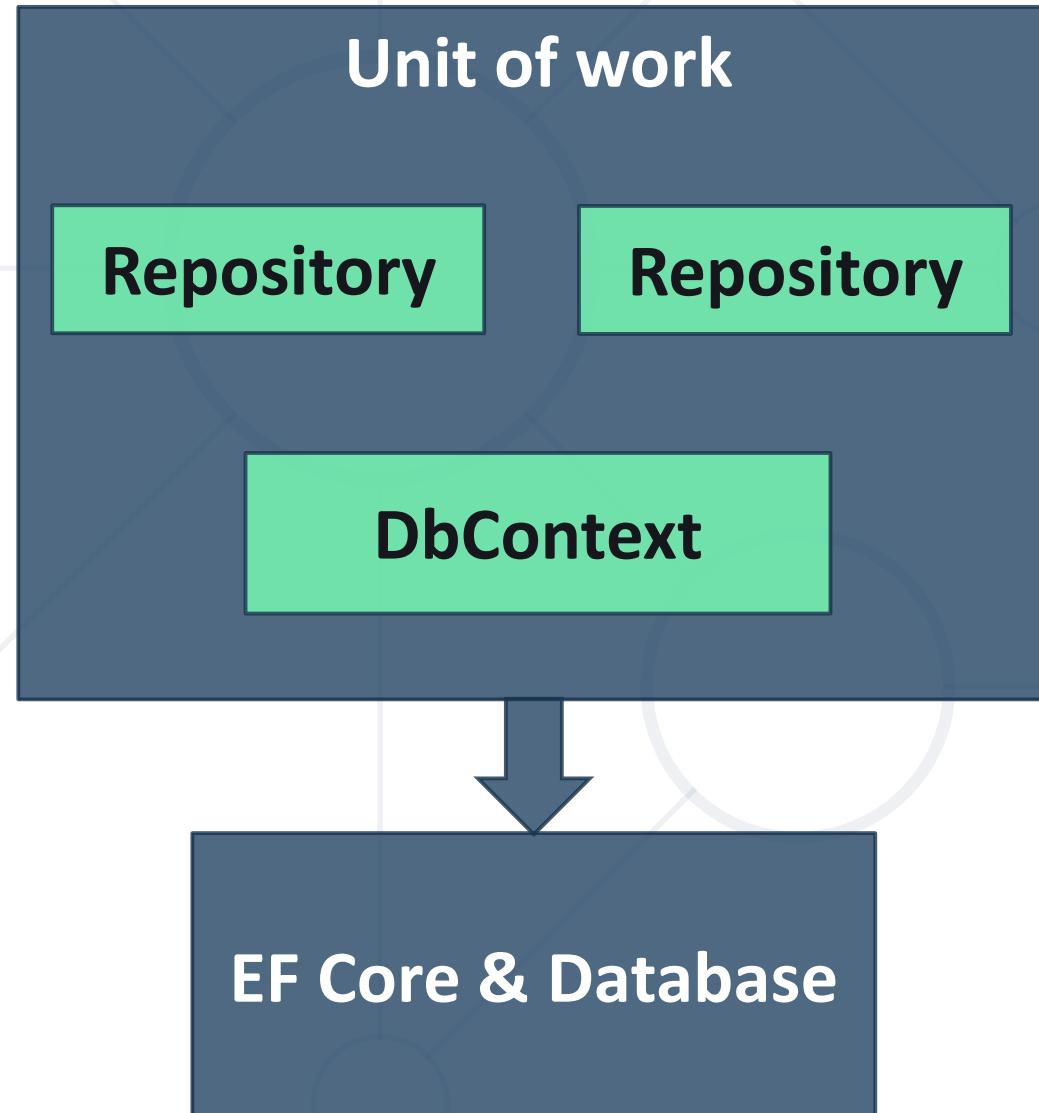


- With repository



- **Serves one purpose** – to make sure that when you use multiple repositories, they share a **single** database context
- With a Unit of Work, you might also choose to implement **Undo / Rollback** functionality
 - When using Entity Framework Core, the recommended approach to undo is to **discard your context** with the changes you are interested in undoing

Unit of Work





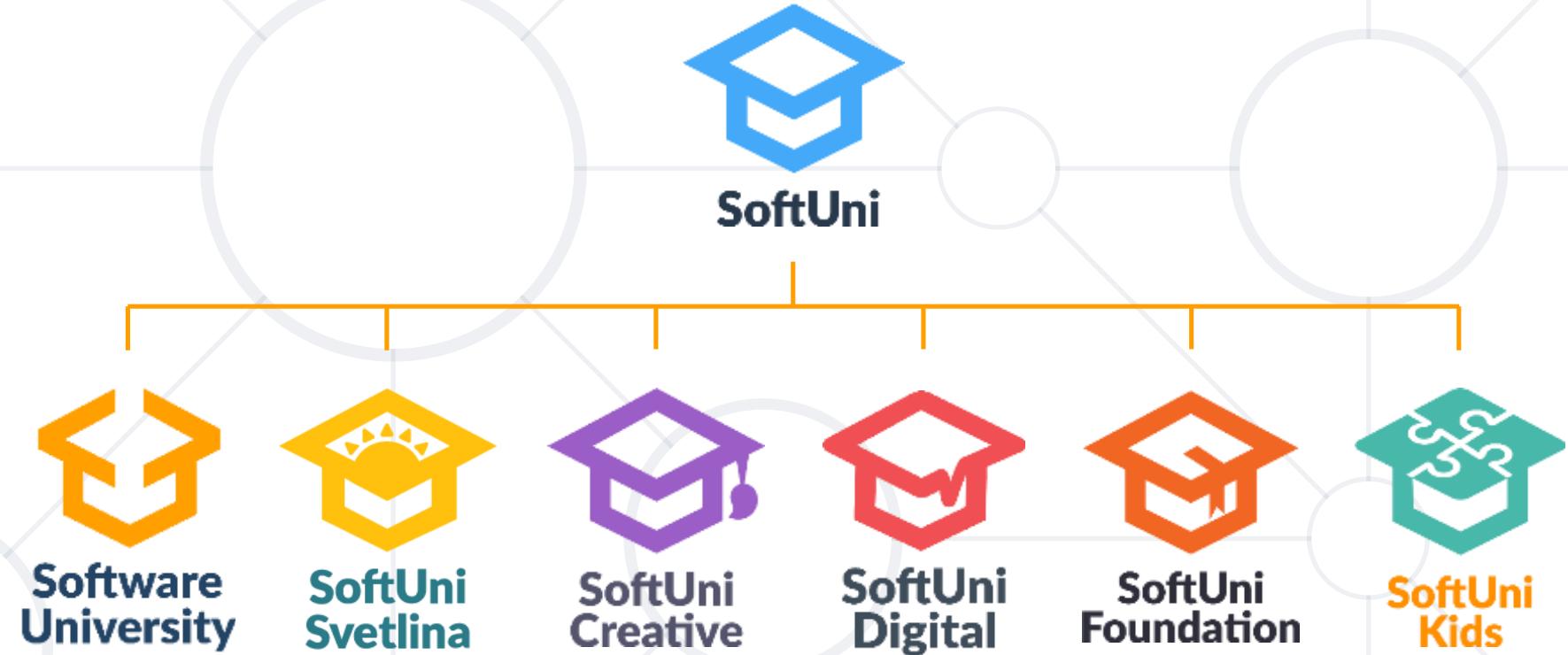
Best Practices and Architecture

Live Demo

- **Project structure** is important as an application is scaled
- Entity Framework Core **performance** can be improved by following certain guidelines
- **Design Patterns** define a common approach to solving certain development problems



Questions?



SoftUni Diamond Partners



SCHWARZ



Coca-Cola HBC
Bulgaria



Postbank

Решения за твоето упре



**SOFTWARE
GROUP**

Bosch.**.IO**



SmartIT



**PHAR
VISION**



CAREERS



INDEAVR
Serving the high achievers



**SUPER
HOSTING
.BG**

Educational Partners



Trainings @ Software University (SoftUni)



- 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
- Software University Forums
 - forum.softuni.bg



Software
University



- 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>



NoSQL and MongoDB

NoSQL vs SQL, MongoDB



SoftUni Team

Technical Trainers



SoftUni



Software University

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

Table of Contents

- Relational and Non-Relational Database
- Database Scalability
- CAP Theorem
- Distributed Systems
- NoSQL database types
- MongoDB Overview
- CRUD Operations

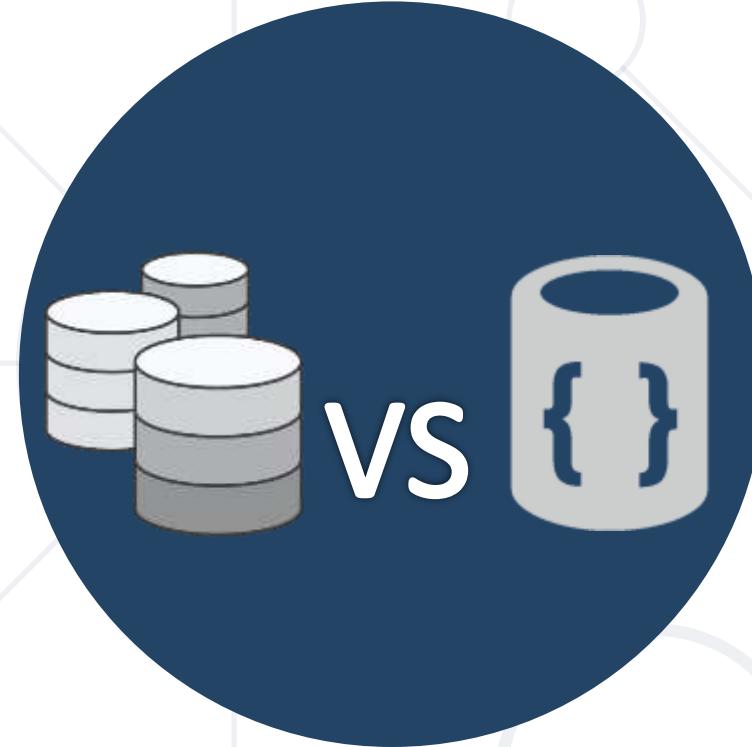


Have a Question?



sli.do

#csharp-db



Relational and Non-Relational Databases

Differences and Examples

Relational Database

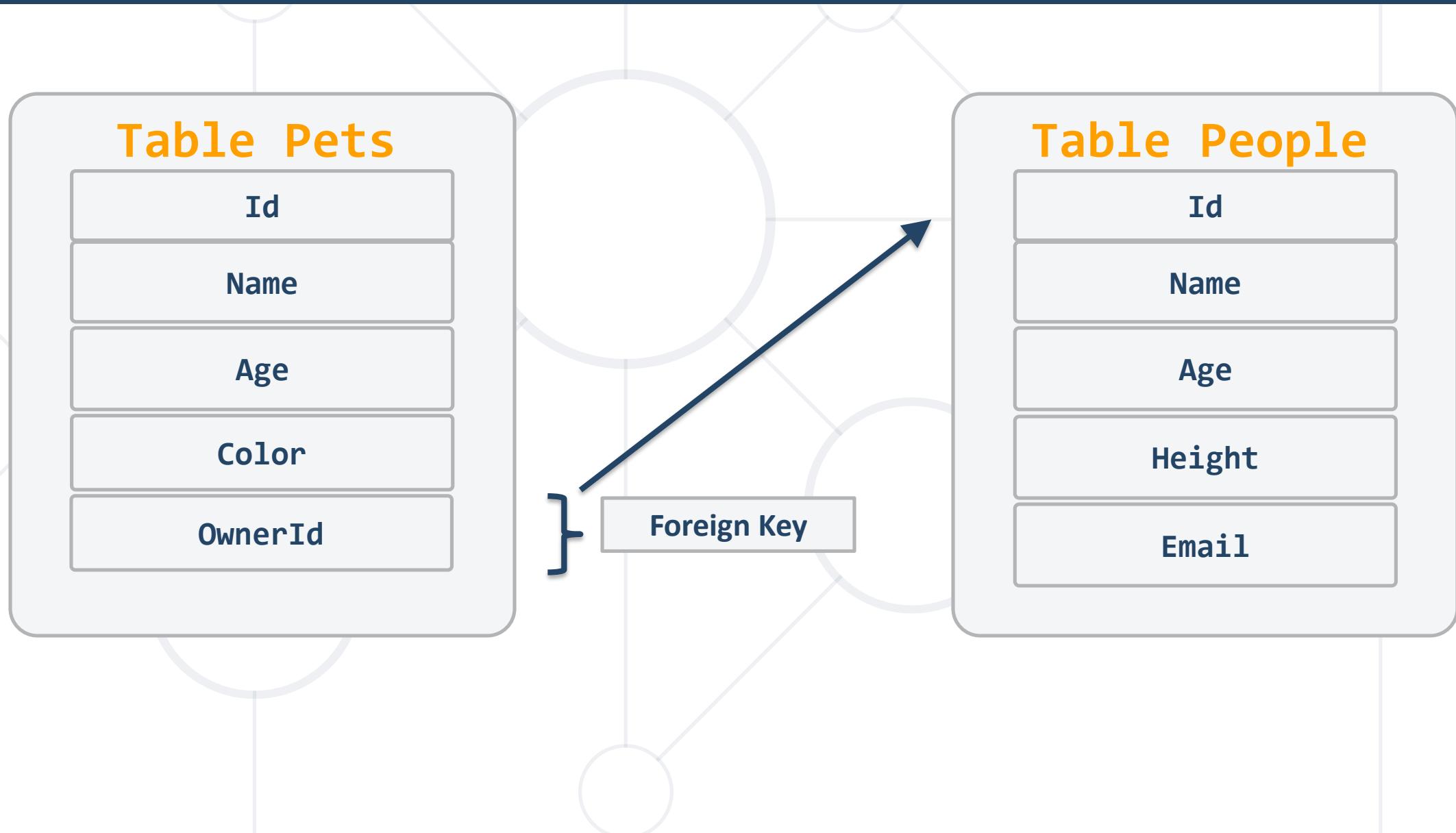
- Organizes data into one or more **tables of columns** and **rows**
- Unique **key** identifying each **row** of data
- Almost all relational databases use **SQL** to **extract** data



```
SELECT * FROM Students
```

- **Relations** between tables are done using **Foreign Keys (FK)**
- Such databases are **Oracle, MySQL, SQL Server**, etc..

Relational Database - Example



Non-relational Database (NoSQL)

- NoSQL Databases are non tabular, and store data differently than relational tables
- Key-value **stores**

```
{  
  "_id": ObjectId("59d3fe7ed81452db0933a871"),  
  "email": "peter@gmail.com",  
  "age": 22  
}
```

- **SQL** query is **not** used in NoSQL systems
- More **scalable** and **provide** superior **performance**



Database Scalability

Database Scalability (1)

Vertical Scaling

(Increase size of instance (RAM ,
CPU etc.))



Horizontal Scaling

(Add more instances)



Database Scalability (2)

- The ability of a system's database to **scale up or down**, depending on the requirements
 - Enables the database to grow to a larger size to support more transactions
 - There are two types of database scalability
 - Vertical Scaling or Scale-up
 - Horizontal Scaling or Scale-out

Vertical Scaling

- Refers to the process of adding more physical resources to the existing database server for improving the performance such as
 - Storage
 - Memory
 - CPU
- Helps in upgrading the capacity of the existing database server

Vertical Scaling Pros and Cons

■ Pros

- It consumes less power
- You need to handle and manage just one system
- Cooling costs are less than horizontal scaling
- Implementation isn't difficult

■ Cons

- Risk of hardware failure which can cause bigger outages
- Limited scope of upgradeability in the future



Horizontal Scaling

- Adds more servers with less RAM and processors
 - The ability to **increase the capacity** by connecting multiple software or hardware entities in a such manner that they function as a single logical unit
 - If a cluster requires more resources to improve its performance and provide high availability, then **the administrator can scale-out by adding more servers** to the cluster

Horizontal Scaling Pros and Cons

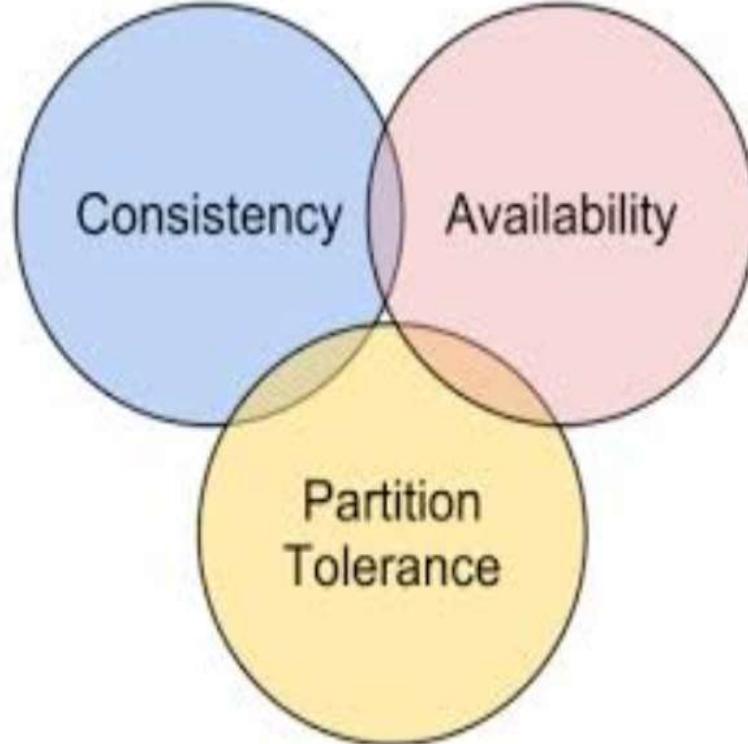
■ Pros

- Easy to upgrade
- Resilience is improved due to the presence of discrete, multiple systems
- Supports linear increases in capacity

■ Cons

- It has a bigger footprint in the Data Center
- Adds complexity to the system
- Introduces data syncing problems
- Dependent on the CAP theorem

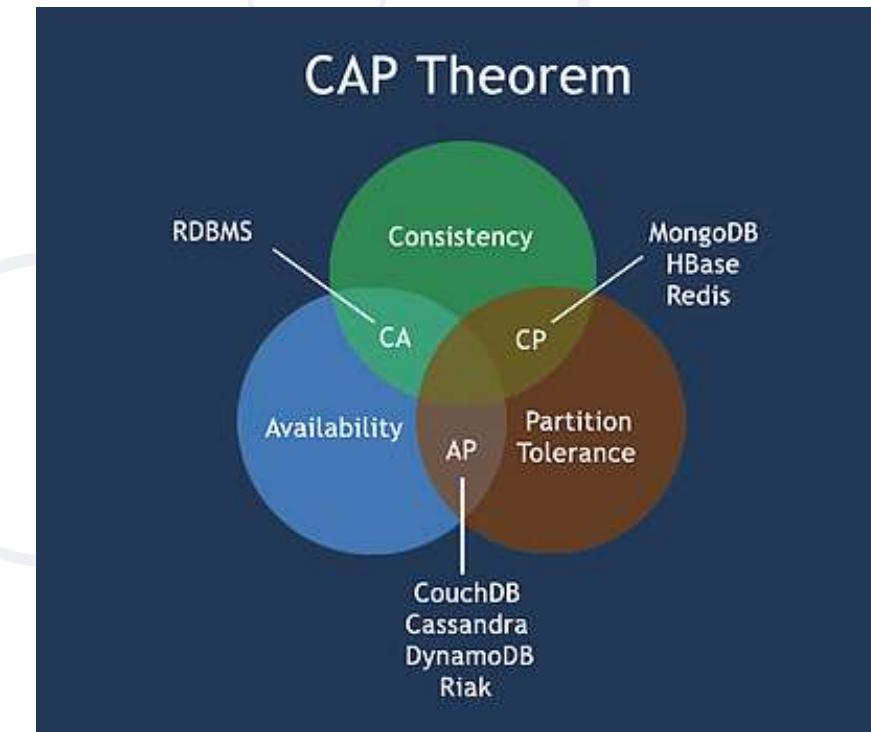


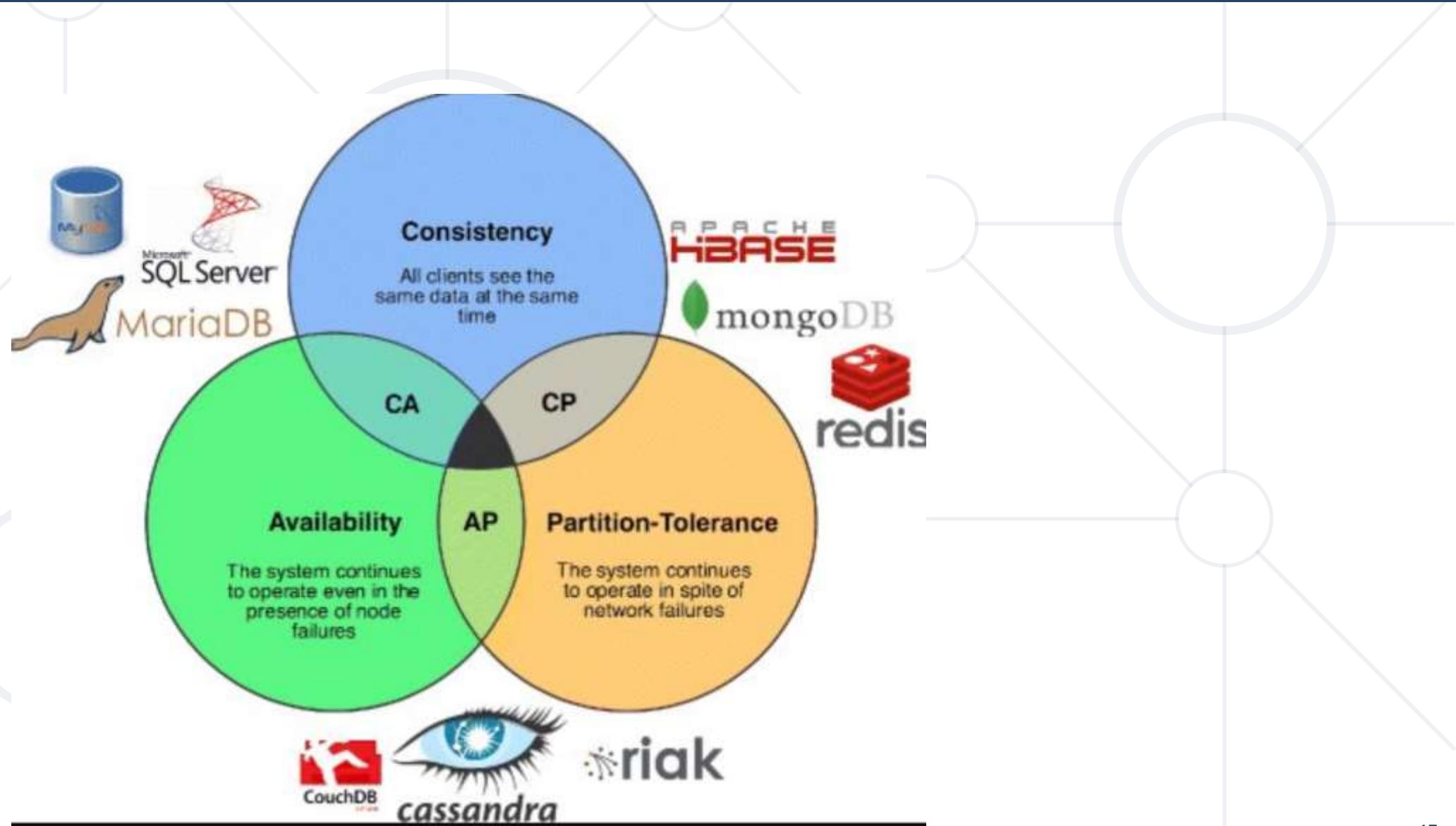


CAP Theorem

What is the CAP Theorem?

- The CAP theorem states that a distributed system can deliver **only two of three** desired characteristics
 - Consistency
 - Availability
 - Partition tolerance





The 'CAP' in the CAP Theorem, Explained (1)

- Consistency
 - All clients see the same data at the same time, no matter which node they connect to
 - Whenever data is written to one node, it must be instantly forwarded or replicated to all the other nodes in the system before the write is deemed 'successful'

The 'CAP' in the CAP Theorem, Explained (2)

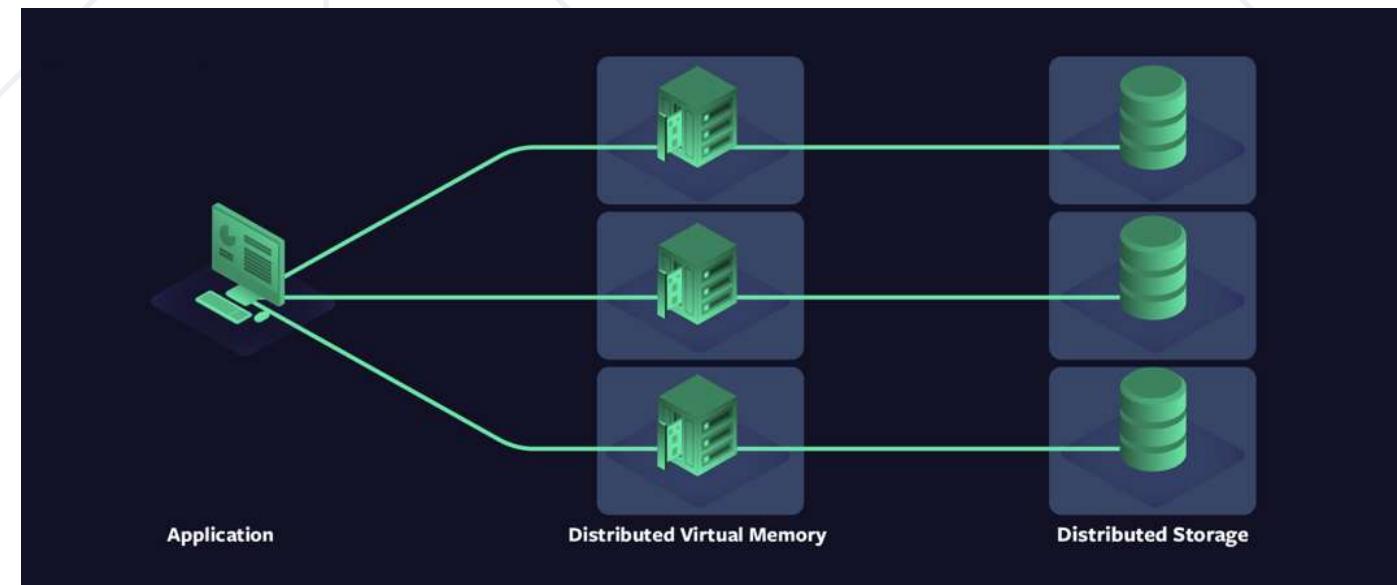
- Availability
 - Any client making a request for data gets a response, even if one or more nodes are down
- Partition tolerance
 - The cluster must continue to work despite any number of communication breakdowns between nodes in the system

Distributed Systems



Distributed Systems

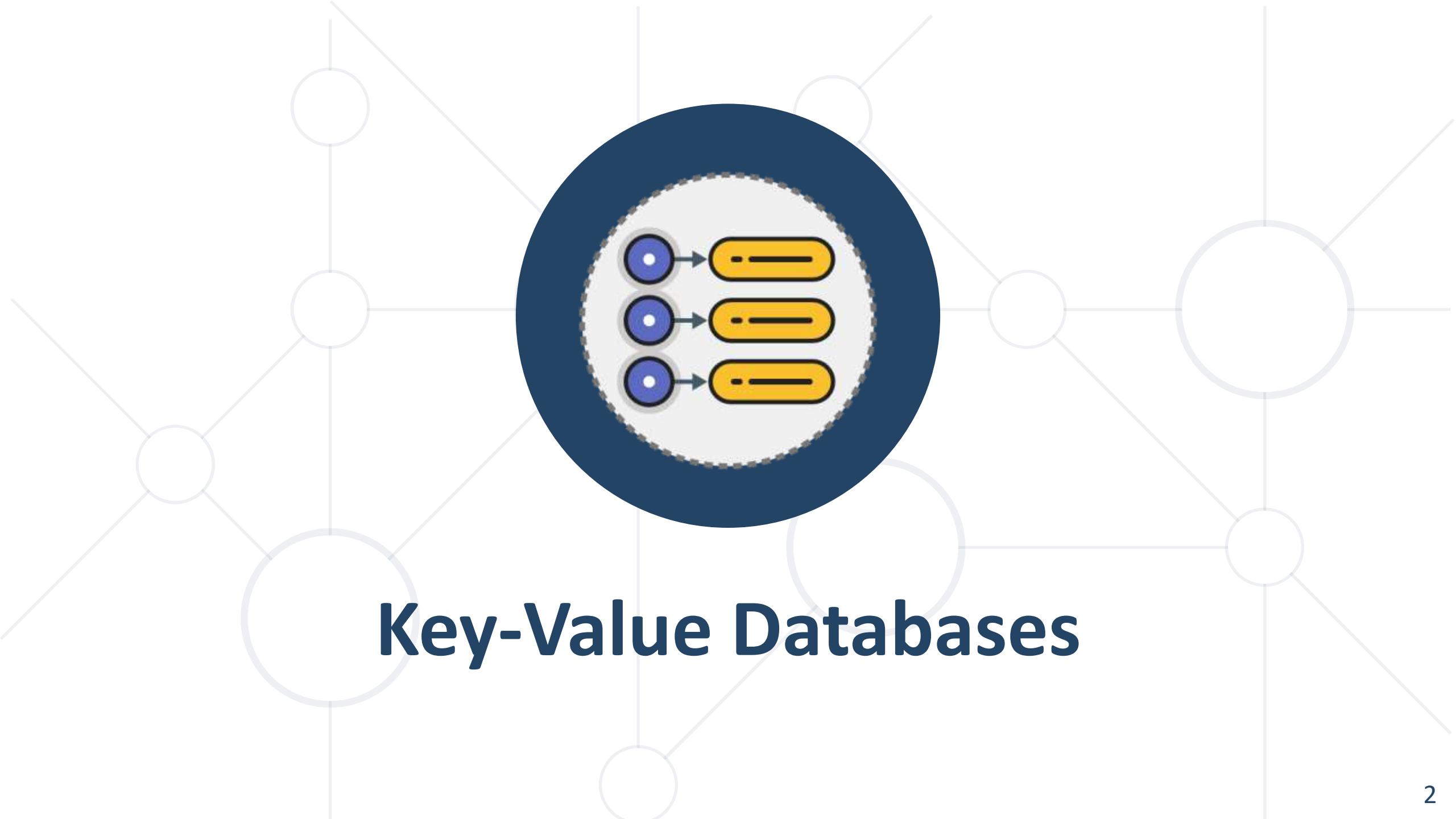
- A network that stores data on more than one physical or virtual machines at the same time
- NoSQL databases are often distributed, and the data is stored on multiple computers



8 Fallacies of Distributed Systems

- The 8 fallacies are
 - The network is reliable
 - Latency is zero
 - Bandwidth is infinite
 - The network is secure
 - Topology doesn't change
 - There is one administrator
 - Transport cost is zero
 - The network is homogeneous



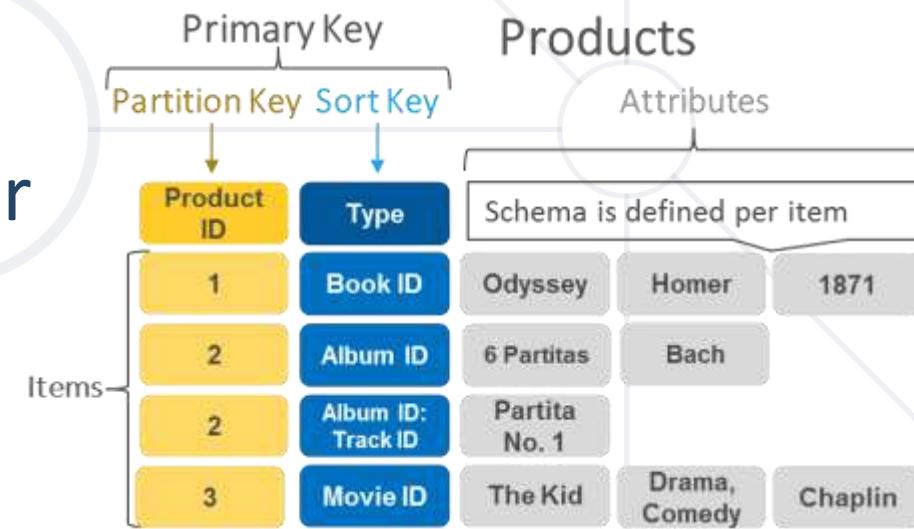


The diagram illustrates a Key-Value Database system. At the center is a dark blue circle containing a white ring. Inside that is another white ring with a dashed border. Within this ring are three blue circles, each connected by an arrow pointing to a yellow horizontal capsule. The capsules contain a short black horizontal line. This central icon is surrounded by a network of light gray circles connected by lines, representing a distributed system.

Key-Value Databases

Key-Value Databases

- Key-value databases work by storing and managing associative arrays
 - Keys serve as a unique identifier to retrieve an associated value
 - Values can be anything from simple objects, like integers or strings, to more complex objects





Document-Oriented Databases

Document-Oriented Databases

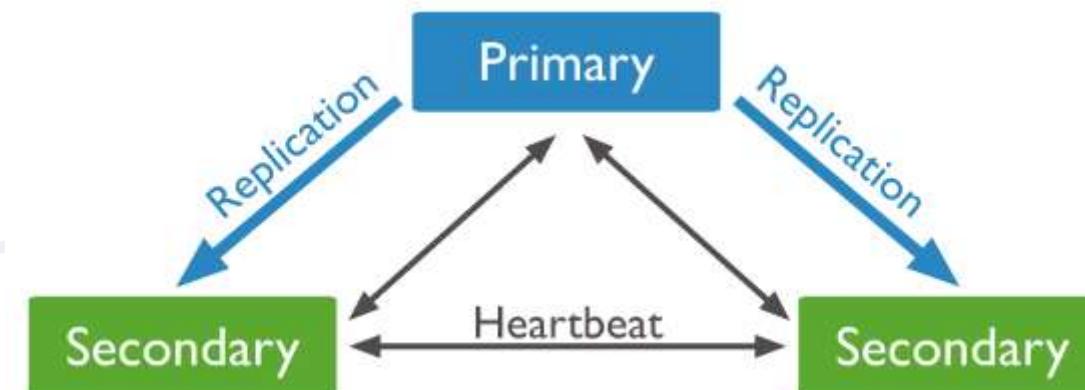
- Document-oriented databases, or document stores, are NoSQL databases that store data in the form of documents
 - Document stores are a type of key-value store
 - Each document has a unique identifier
 - Document itself serves as the value
 - Usually stored as JSON, XML, Proto-Buff, etc.

```
{  
    "FirstName": "Bob",  
    "Address": "5 Oak St.",  
    "Hobby": "sailing"  
}
```



Replication

- A **replica** set is a group of mongod instances that maintain the same data set
- If the primary is unavailable, an eligible secondary will hold an election to elect itself the new primary
- All reads and writes happen from the primary (configurable)

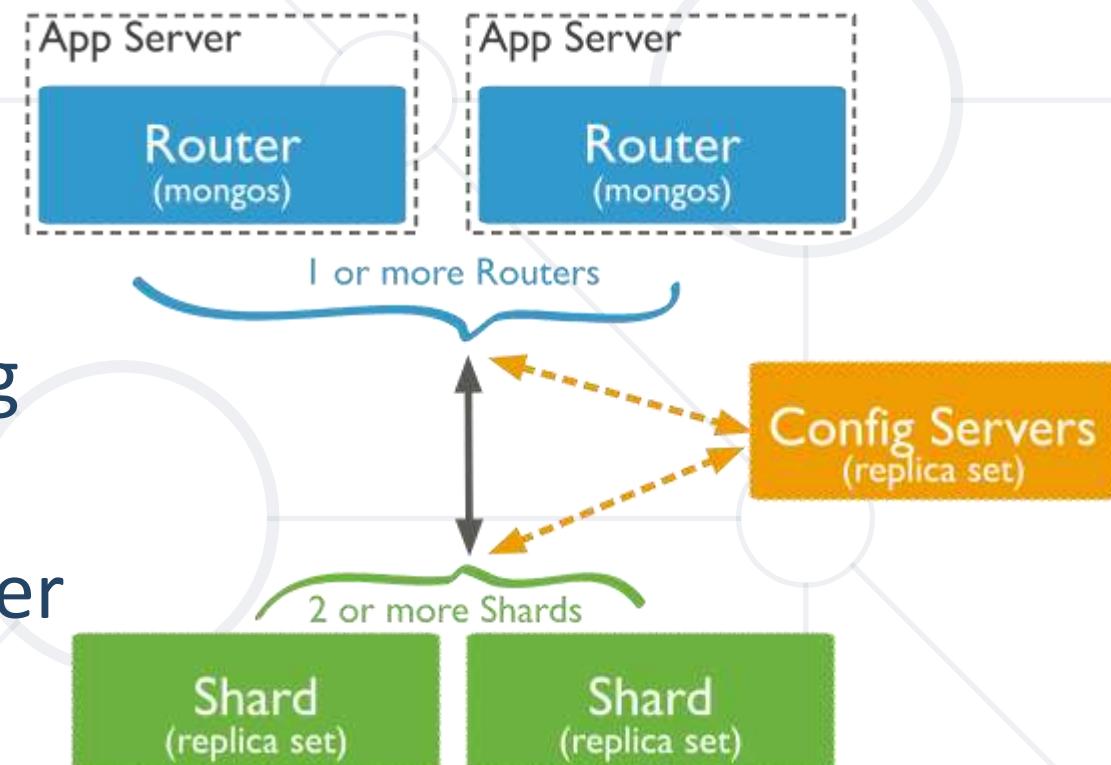


Sharding

- **Sharding** is a method for distributing data across multiple machines
- MongoDB uses sharding to support deployments with very large data sets and high throughput operations
- Database systems with large data sets or high throughput applications can challenge the capacity of a single server

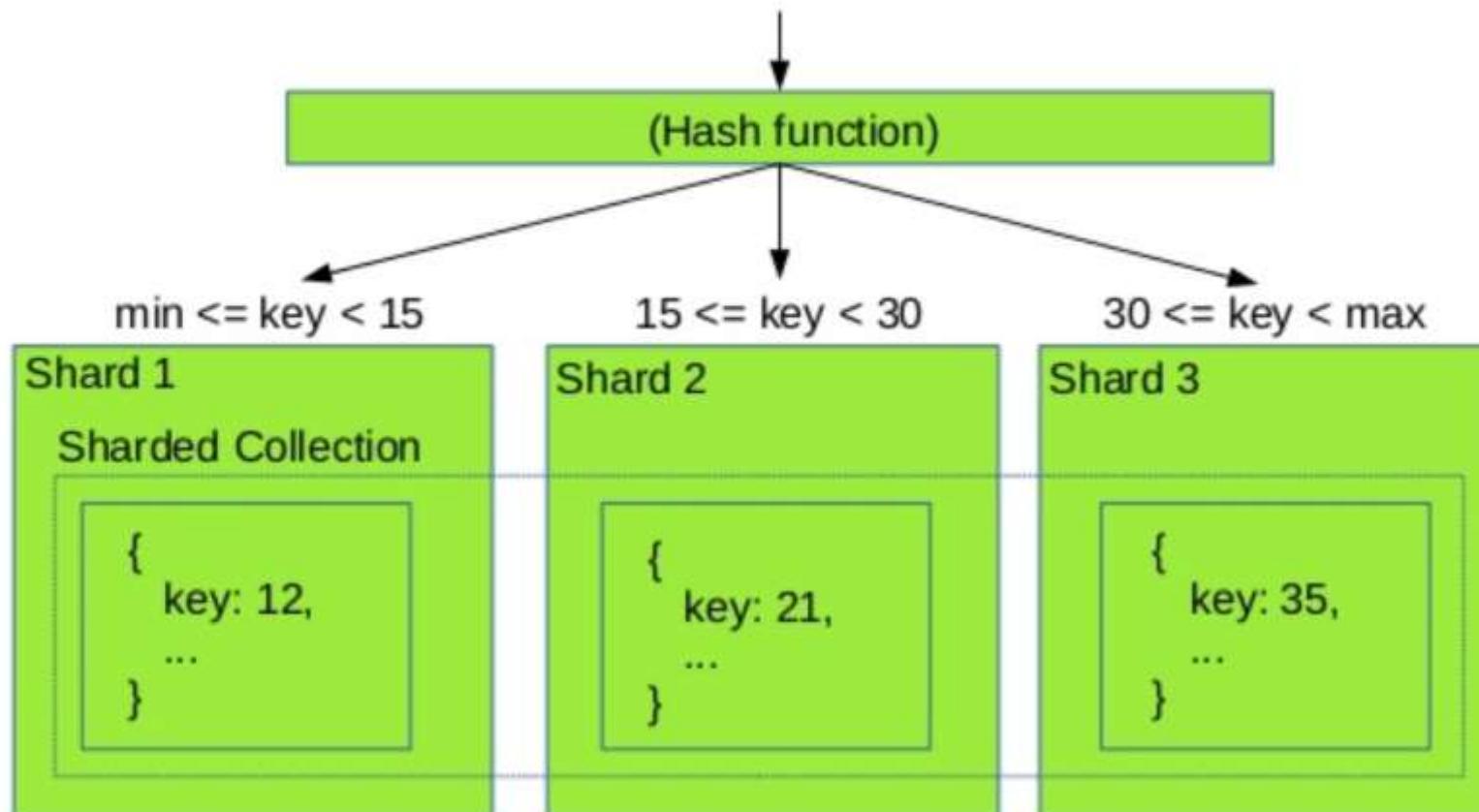
Sharded Cluster

- A MongoDB **sharded cluster** consists of the following components
 - **Shard** – each contains a subset of the sharded data
 - **Mongos** - a query router, providing an interface between client applications and the sharded cluster
 - **Config Servers** - store metadata and configuration settings for the cluster

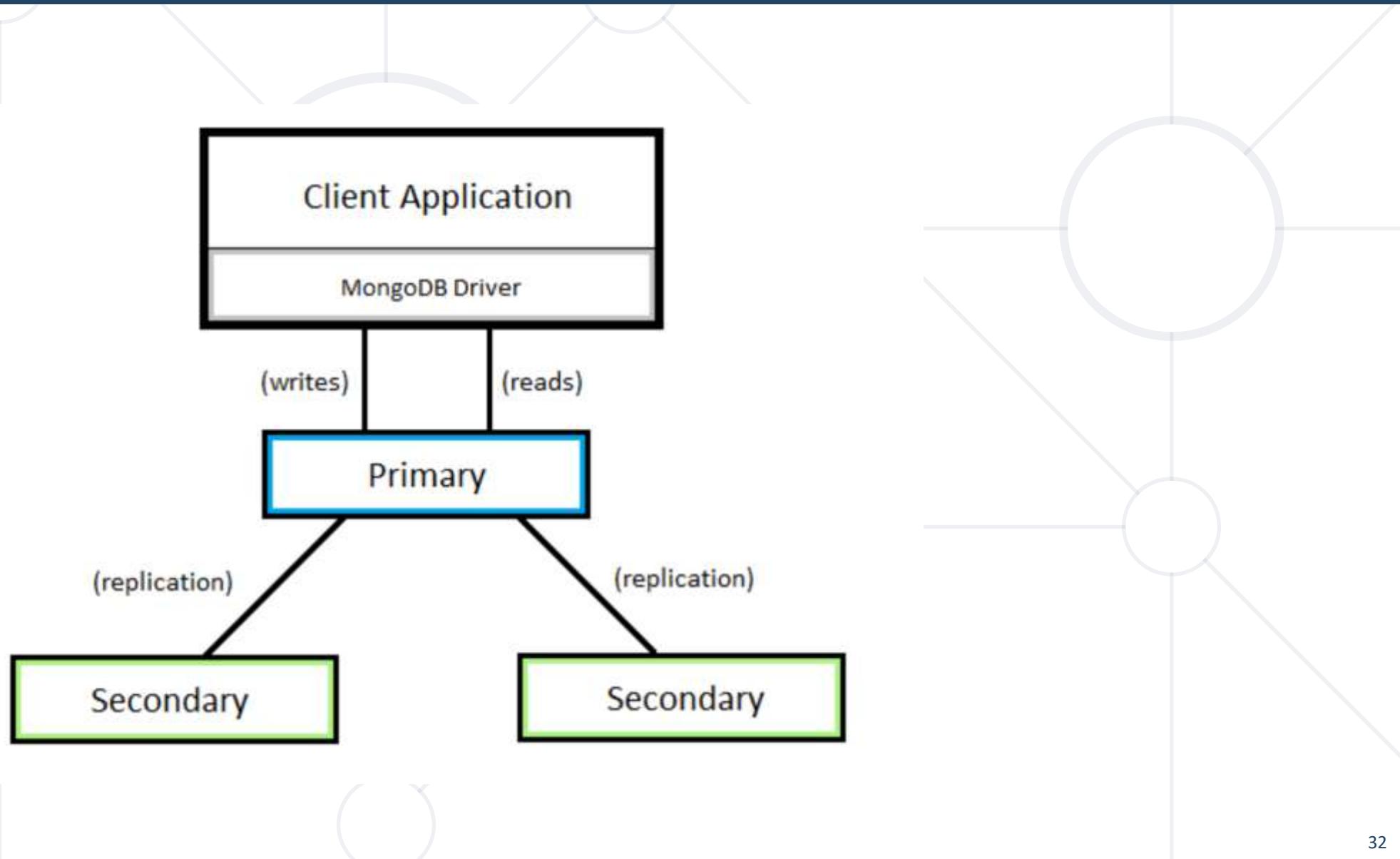


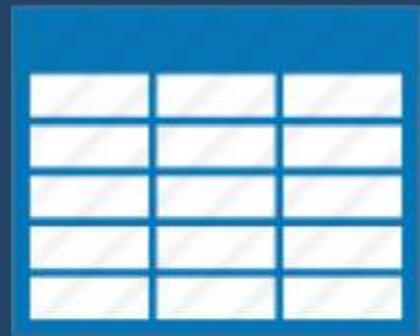
MongoDB

Shard key selection



Replication





Columnar Databases

Columnar Databases (1)

- Columnar databases, are database systems that store data in columns
 - Each column is stored in a separate file or region in the system's storage
 - Examples of columnar databases are Cassandra, Hbase, Redshift, etc.

Row-oriented (1)

name	age	sex	zipcode
thomas	18	male	1416
martin	33	male	1645
bob	25	male	1613

Column-oriented (2)

name	age	sex	zipcode
thomas	18	male	1416
martin	33	male	1645
bob	25	male	1613

Columnar Databases (2)

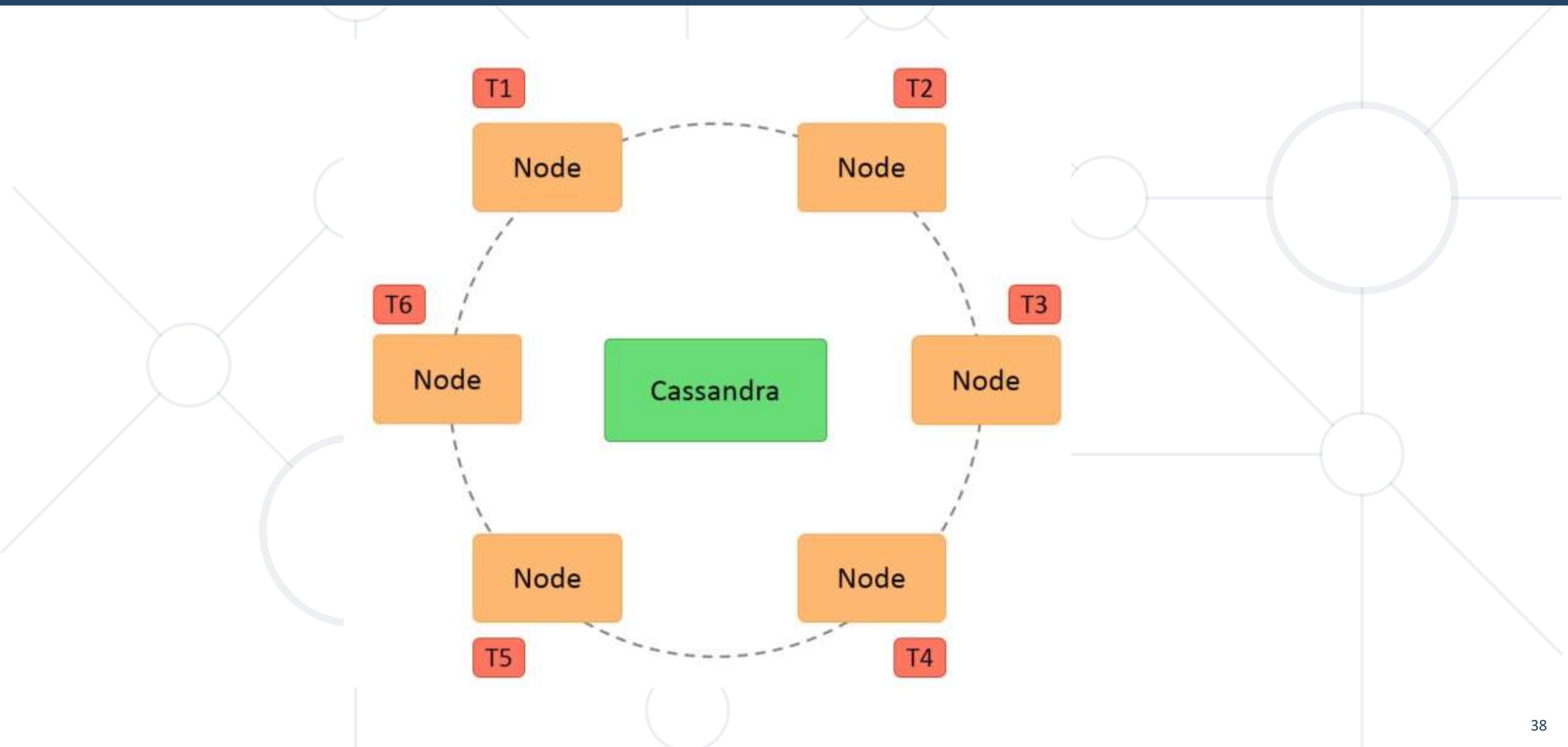
- Key benefits of column store databases include faster performance in load, search, and aggregate functions
- Column-oriented organizations are more efficient when an aggregate needs to be computed over many rows but only for a notably smaller subset of all columns of data
- Not efficient when many columns of a row are required at the same time

Columnar Databases (3)

AuthorProfile			
Mahesh	Gender	Expertise	Rank
Male	ADO.NET, C#, GDI+	102	
03182019	03182019	03182019	
David			
David	Gender	Book	
Male	AWS Developer's Guide		
03202019	03202019		
Allen			
Allen	City	Book	Rank
London	Azure Quickstart	89	
04201019	04201019	04201019	

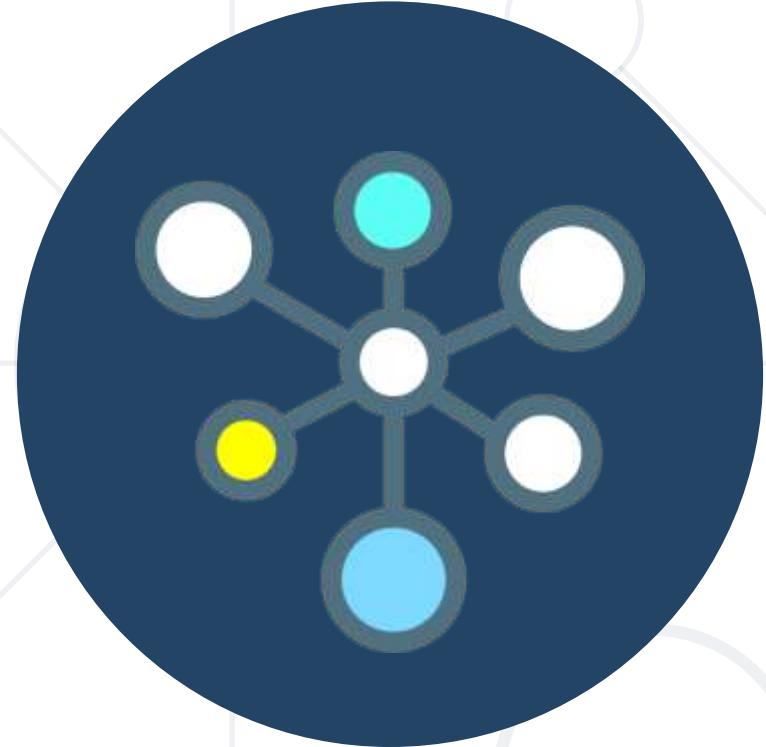
- Generally considered AP (in CAP)
- Every node in the cluster has the same role. There is no single point of failure.
- Data is distributed across the cluster (so each node contains different data)
- Failed nodes can be replaced with no downtime.
- Eventually consistent (configurable)
- Uses CQL for queries

Cassandra (2)



Cassandra Usage

- **Discord** switched to Cassandra to store billions of messages from MongoDB in November, 2015
- **Netflix** uses Cassandra as their back-end database for their streaming services
- **Apple** uses 100,000 Cassandra nodes
- **Uber** uses Cassandra to store around 10,000 features
- Many more applications



Graph Database

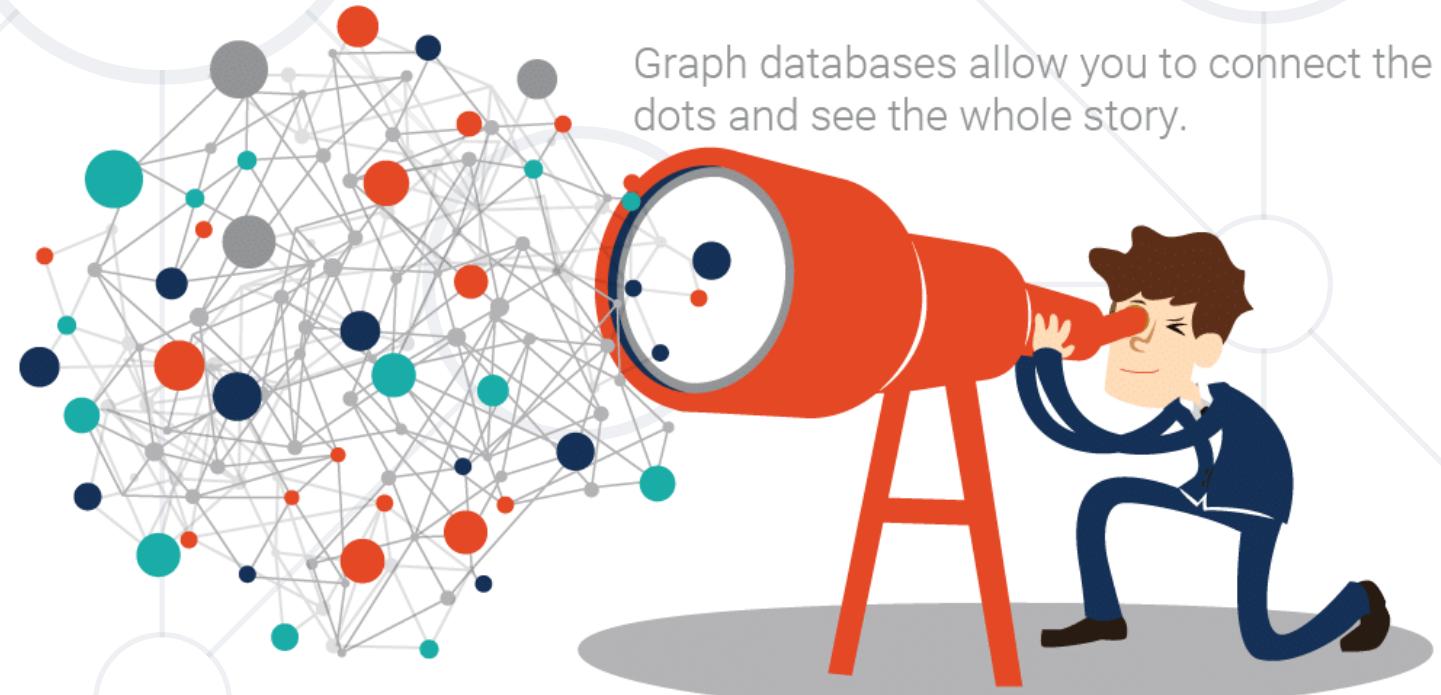
What is a Graph Database? (1)

- Allow simple and fast retrieval of complex hierarchical structures that are difficult to model in relational systems
- No universal query language is present for graph databases (like SQL). Each database has own implementation of queries



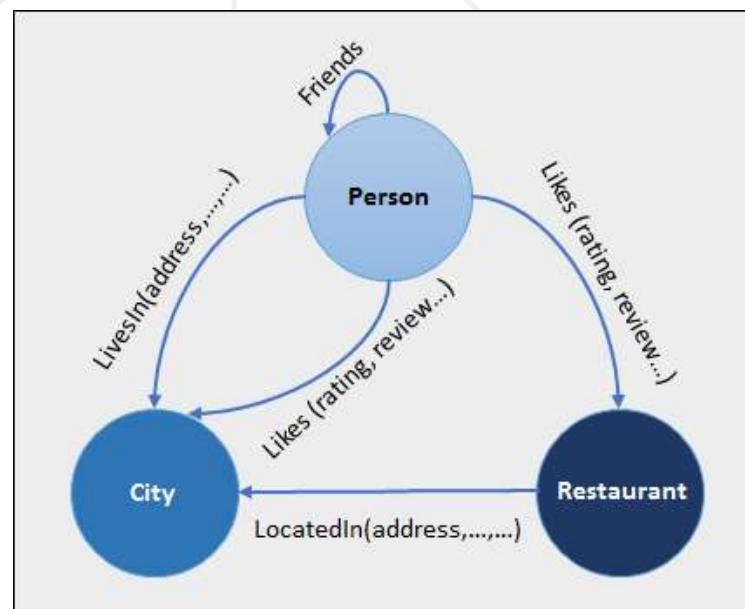
What is a Graph Database? (2)

- A graph database contains a collection of nodes and edges
 - A node represents an object
 - An edge represents the connection or relationship between two objects



What is a Graph Database? (3)

- Each node is identified by a unique identifier that expresses key-value pairs
- Each edge is defined by a unique identifier that details a starting or ending node, along with a set of properties





MongoDB Overview

Installation, Configuration, Startup

What is MongoDB?

- MongoDB is a **document database**
- It stores data in flexible, **BSON** documents
- The document model maps to the objects in the application code, making data easy to work with
- MongoDB is a **distributed database** at its core



Install MongoDB

- Download from: <https://www.mongodb.com/download-center>
- When **installed**, MongoDB needs a **driver**
 - One to use with Node.js, .NET, Java, etc..
 - MongoDB C#/.NET driver:
<https://docs.mongodb.com/drivers/csharp>

Working with MongoDB GUI

- Choose one of the many
- For example
 - Robo 3T → <https://robomongo.org/download>
 - NoSQLBooster → <https://nosqlbooster.com>
 - Compass → <https://www.mongodb.com/products/compass>

Working with MongoDB Shell Client

- Start the shell from a **CLI**

- Type the command **mongo**

```
show dbs
```

Shows **all** databases
in the data **folder**

```
use mytestdb
```

```
db.mycollection.insert({"name": "George"})
```

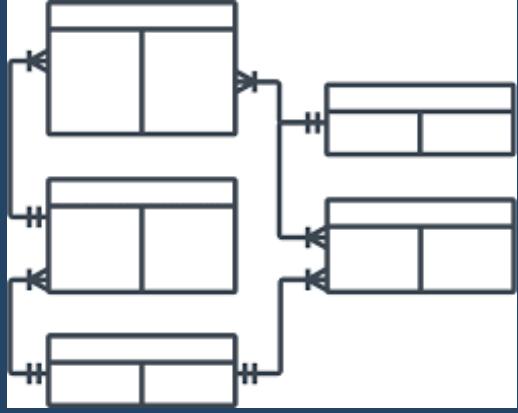
```
db.mycollection.find({"name": " George"})
```

```
db.mycollection.find({})
```

Gets **all** entries in
the database

- Additional information at

<https://docs.mongodb.com/manual/reference/mongo-shell/>



CRUD Operations

Connect to MongoDB

- To **connect** to a MongoDB cluster, use the connection string for your cluster

```
using MongoDB.Bson;
using MongoDB.Driver;
...
var client = new MongoClient(
    "mongodb+srv://<username>:<password>@<cluster-address>/test?w=majority"
);
var database = client.GetDatabase("Example");
var collection = database.GetCollection<Interactions>("Interactions");
```

Select

- To **select** a document use Linq

```
var result = IMongoCollectionExtensions  
    .AsQueryable(collection)  
    .FirstOrDefault(s => s.SiteName == "Example");
```

- **FindOneAndUpdate()**

```
var update = MongoDB.Driver.Builders.Update.Set(s => s.SiteName,  
    "New Example");  
  
collection.FindOneAndUpdate(s => s.SiteName == "Example",  
    update);
```

Delete and Insert

- **DeleteOne()**

- Deletes the first document that meets the filter

```
collection.DeleteOne(e => e.Name == "Example");
```

- **InsertOne()**

- Inserts a new document

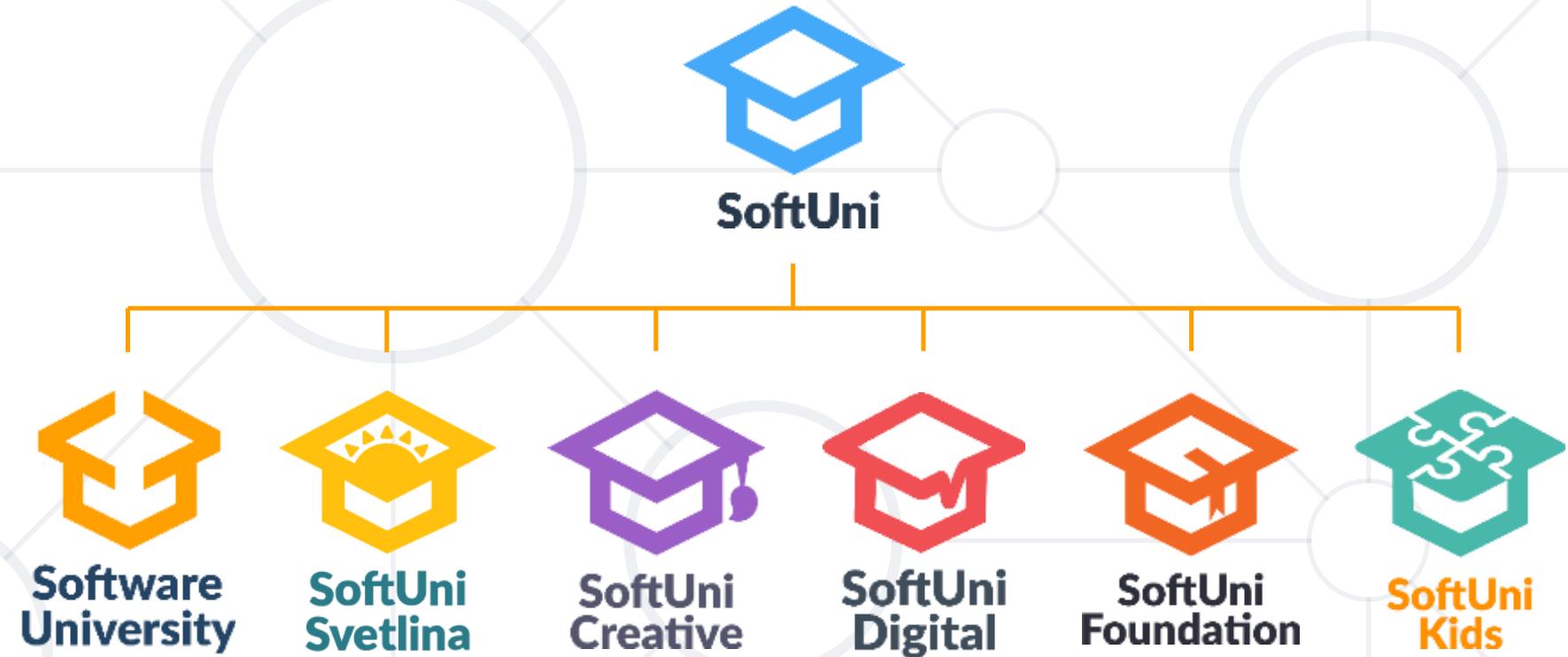
```
collection.InsertOne(newItem);
```

Summary

- NoSQL
- Ability of a system's DB to **scale up or down**
- CAP Theorem
- Distributed Systems
- Key-value, Document-oriented, Columnar and **Graph** DBs
- MongoDB Overview
- CRUD Operations



Questions?



SoftUni Diamond Partners



SCHWARZ



Coca-Cola HBC
Bulgaria



Postbank

Решения за твоето упре



**SOFTWARE
GROUP**

Bosch.**.IO**



SmartIT



**PHAR
VISION**



INDEAVR
Serving the high achievers



**SUPER
HOSTING
.BG**

Educational Partners



Trainings @ Software University (SoftUni)



- 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
- Software University Forums
 - forum.softuni.bg



Software
University



- 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>

