GDSC - SHA
Entity Framework Core

Collected EF Core Sessions

By Youssef Adel

```
                     _/\__
               ---==/    \\
         ___  ___   |.    \|\
        | __|| __|  |  )   \\\
        | _| | _|   \_/ |  //|\\
        |___||_|       /   \\\/\\
```

ORM

- Is a technique that lets you query the data from the database using Csharp oop paradigm.

Dapper

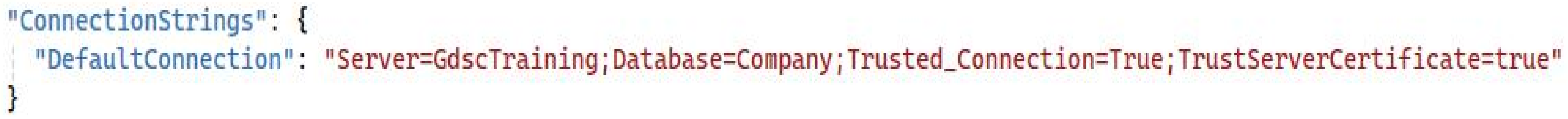
NHIBERNATE

Entity Framework Core

Connection String

- Is a string that specifies information about a data source and the means of connecting to it.

- It is the value that connects your app to database.



```
"ConnectionStrings": {
  "DefaultConnection": "Server=GdscTraining;Database=Company;Trusted_Connection=True;TrustServerCertificate=true"
}
```

```
                     _/\__
               ---==/    \\
         ___  ___   |.    \|\
        | __|| __|  |  )    \\\
        | _| | _|   \_/ |  //|\\
        |___||_|       /   \\\/\\
```

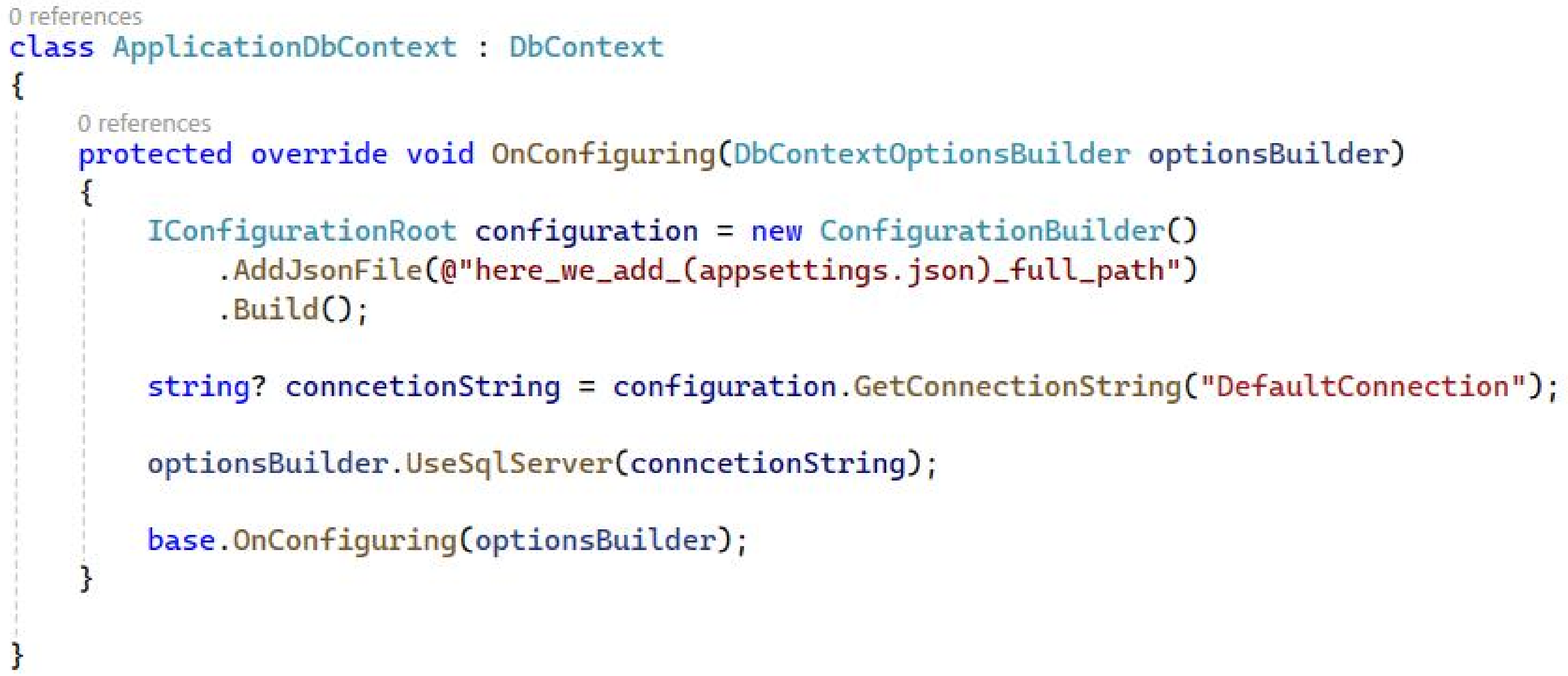
How to use EFcore in Your app?

1. First you need to install packages
   - Microsoft.EntityFrameworkCore
   - Microsoft.EntityFrameworkCore.Tools
   - Microsoft.EntityFrameworkCore.SqlServer
   - Microsoft.Extensions.Configuration
   - Microsoft.Extensions.Configuration.json

2. ApplicationDbContext class and inherit from DbContext.

How to use EFcore in Your app?

3. Inside ApplicationDbContext override method OnConfiguring

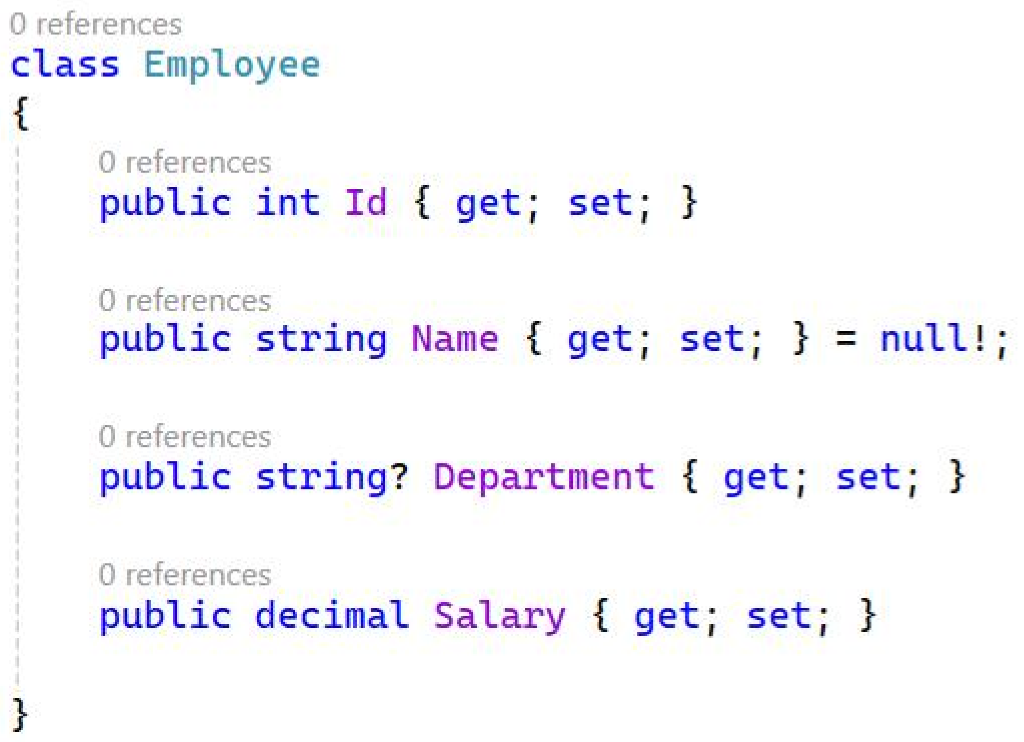


```
0 references
class ApplicationDbContext : DbContext
{
    0 references
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        IConfigurationRoot configuration = new ConfigurationBuilder()
            .AddJsonFile(@"here_we_add_(appsettings.json)_full_path")
            .Build();

        string? conncetionString = configuration.GetConnectionString("DefaultConnection");

        optionsBuilder.UseSqlServer(conncetionString);

        base.OnConfiguring(optionsBuilder);
    }

}
```

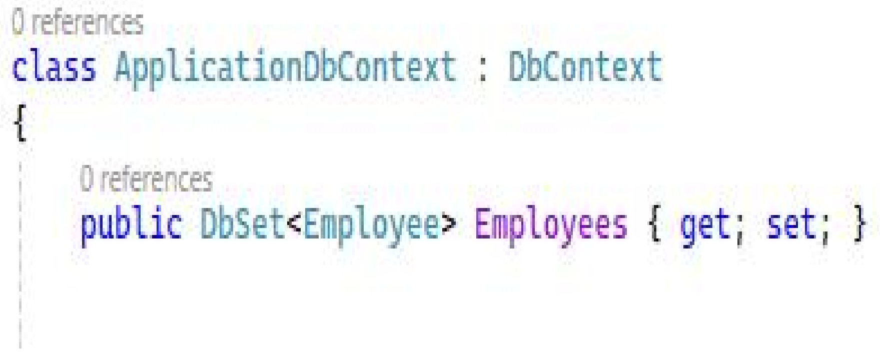
How to use EFcore in Your app?

4. Create database entities classes.

5. Create a DbSet<entityType> property inside ApplicationDbContext.



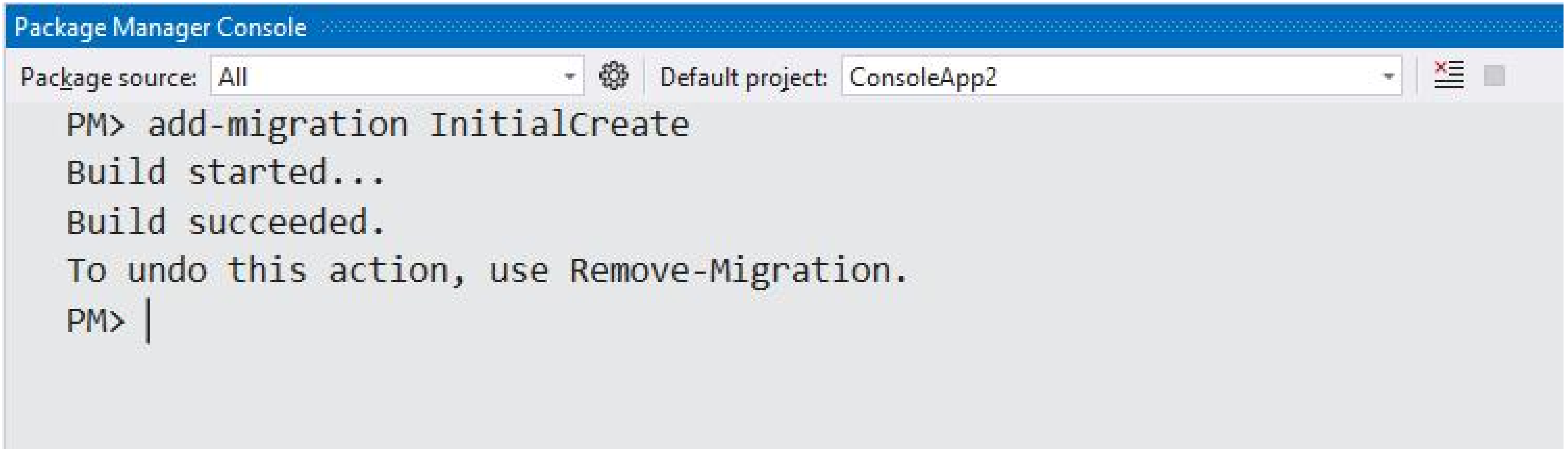
```
0 references
class Employee
{
    0 references
    public int Id { get; set; }

    0 references
    public string Name { get; set; } = null!;

    0 references
    public string? Department { get; set; }

    0 references
    public decimal Salary { get; set; }

}
```



```
0 references
class ApplicationDbContext : DbContext
{
    0 references
    public DbSet<Employee> Employees { get; set; }
```

How to use EFcore in Your app?

6. Add new migration InitialCreate



Package source: All

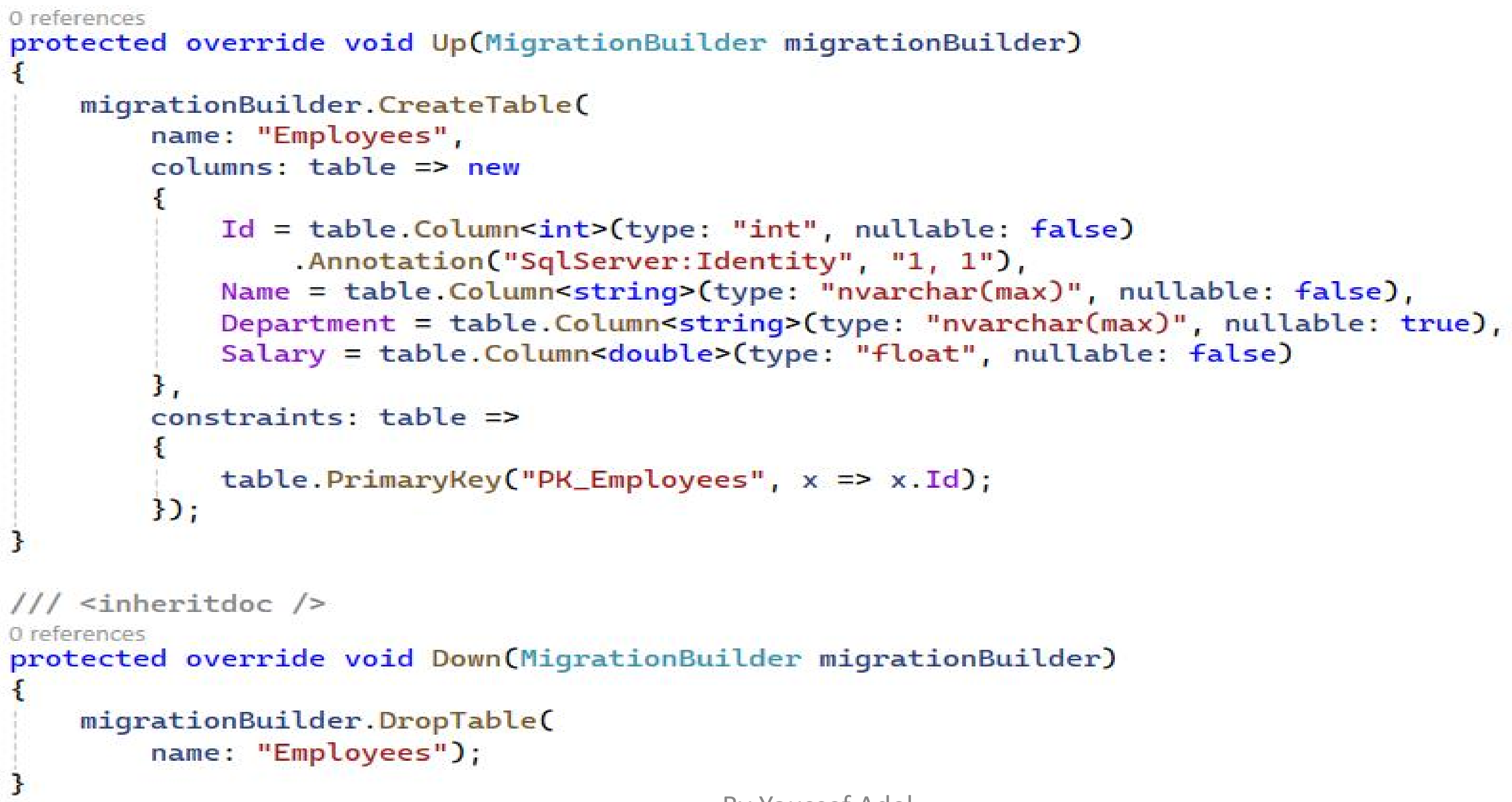
Default project: ConsoleApp2

```
PM> add-migration InitialCreate
Build started...
Build succeeded.
To undo this action, use Remove-Migration.
PM>
```

```
                     _/\__
               ---==/    \\
         ___  ___   |.    \|\
        | __|| __|  |  )   \\\
        | _| | _|   \_/ |  //|\\
        |___||_|       /   \\\/\\
```

How to use EFcore in Your app?

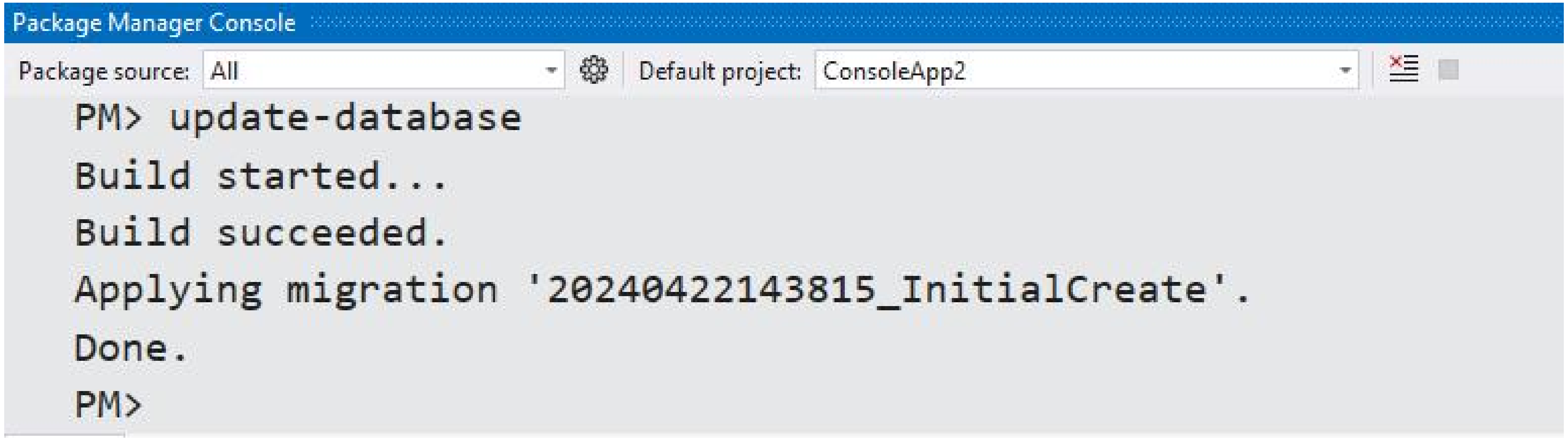
7. Read the migration



```
0 references
protected override void Up(MigrationBuilder migrationBuilder)
{
    migrationBuilder.CreateTable(
        name: "Employees",
        columns: table => new
        {
            Id = table.Column<int>(type: "int", nullable: false)
                .Annotation("SqlServer:Identity", "1, 1"),
            Name = table.Column<string>(type: "nvarchar(max)", nullable: false),
            Department = table.Column<string>(type: "nvarchar(max)", nullable: true),
            Salary = table.Column<double>(type: "float", nullable: false)
        },
        constraints: table =>
        {
            table.PrimaryKey("PK_Employees", x => x.Id);
        });
}

/// <inheritdoc />
0 references
protected override void Down(MigrationBuilder migrationBuilder)
{
    migrationBuilder.DropTable(
        name: "Employees");
}
```

How to use EFcore in Your app?

8. Apply your changes to the database



```
PM> update-database
Build started...
Build succeeded.
Applying migration '20240422143815_InitialCreate'.
Done.
PM>
```

```
                     _/\__
               ---==/    \\
         ___  ___   |.    \|\
        | __|| __|  |  )    \\\
        | _| | _|   \_/ |  //|\\
        |___||_|       /   \\\/\\
```

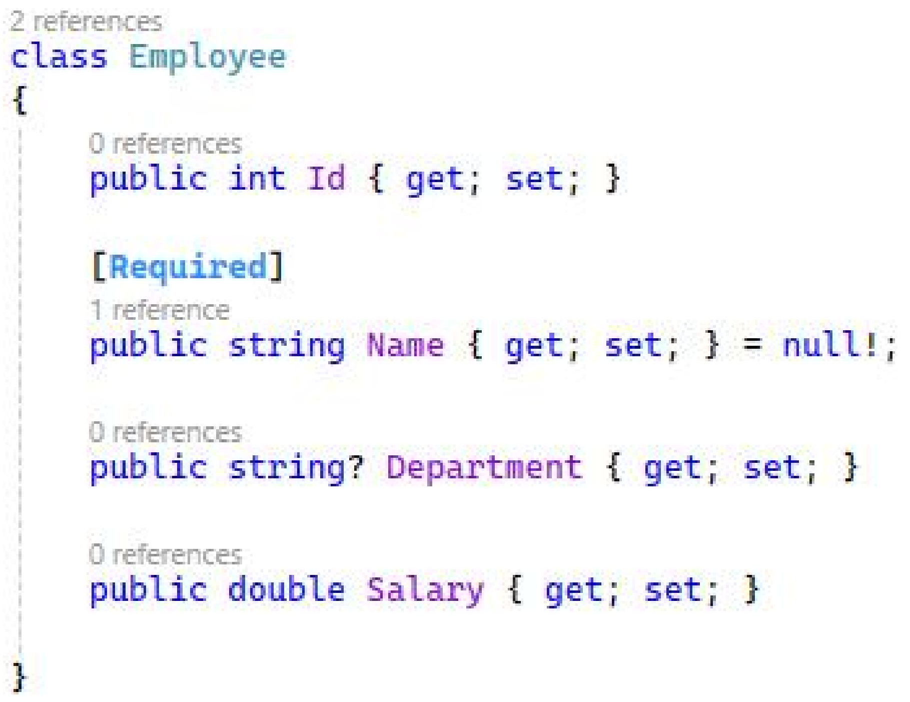
Most used commands

- Add-Migration migration_name → adds new migration.
- Remove-Migration → remove the latest migration.
- Update-Database → applies migrations on the database.
- Update-Database migration_name → returns the database to specified migration.
- Update-Database 0 → remove all migrations

Data Annotation VS Fluent API

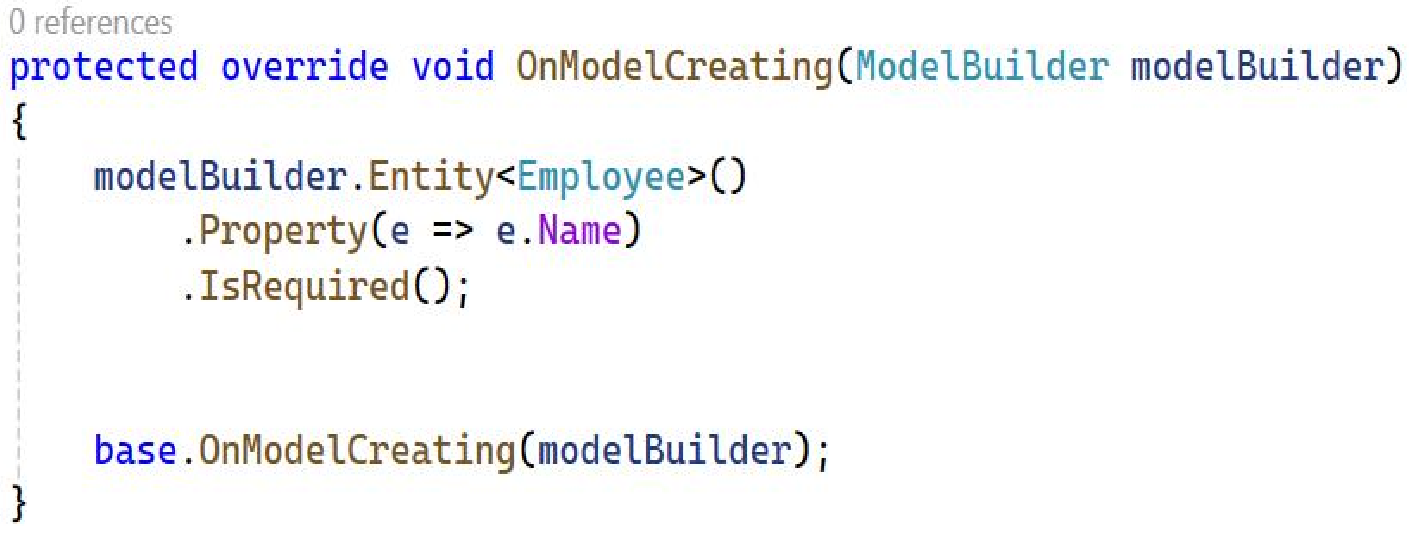
- To make the Column required.

Using Data Annotations



```
2 references
class Employee
{
    0 references
    public int Id { get; set; }

    [Required]
    1 reference
    public string Name { get; set; } = null!;

    0 references
    public string? Department { get; set; }

    0 references
    public double Salary { get; set; }

}
```

Using Fluent API

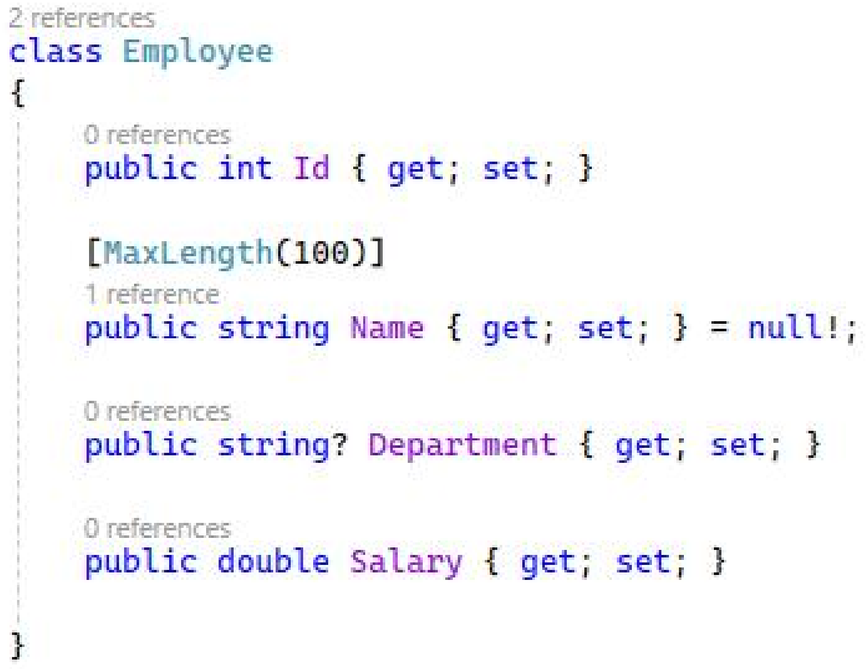


```
0 references
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Employee>()
        .Property(e => e.Name)
        .IsRequired();


    base.OnModelCreating(modelBuilder);
}
```

Data Annotation VS Fluent API

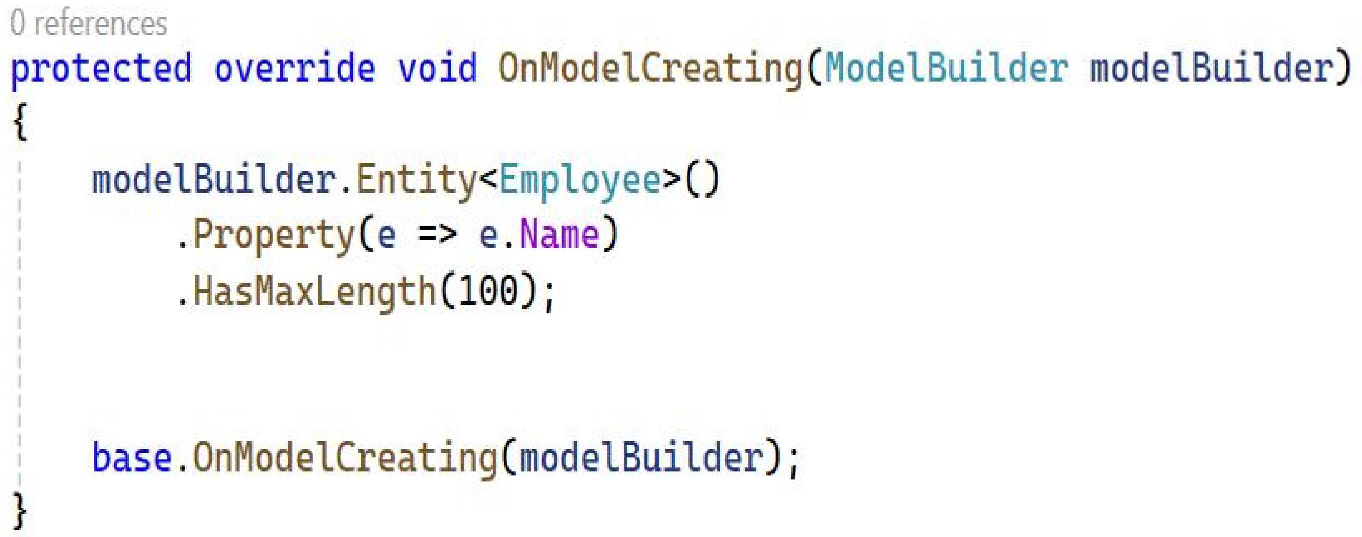
- Set Maximum Length

Using Data Annotations



```
2 references
class Employee
{
    0 references
    public int Id { get; set; }

    [MaxLength(100)]
    1 reference
    public string Name { get; set; } = null!;

    0 references
    public string? Department { get; set; }

    0 references
    public double Salary { get; set; }

}
```

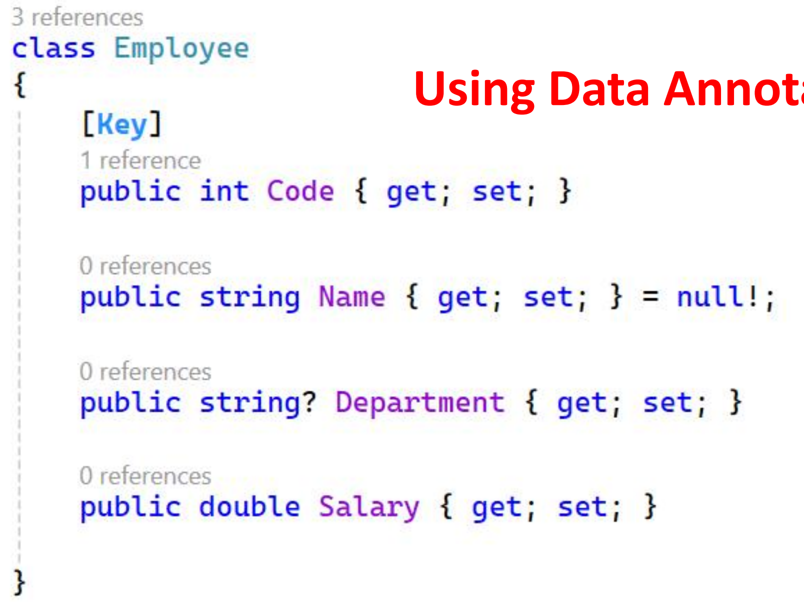
Using Fluent API



```
0 references
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Employee>()
        .Property(e => e.Name)
        .HasMaxLength(100);


    base.OnModelCreating(modelBuilder);
}
```

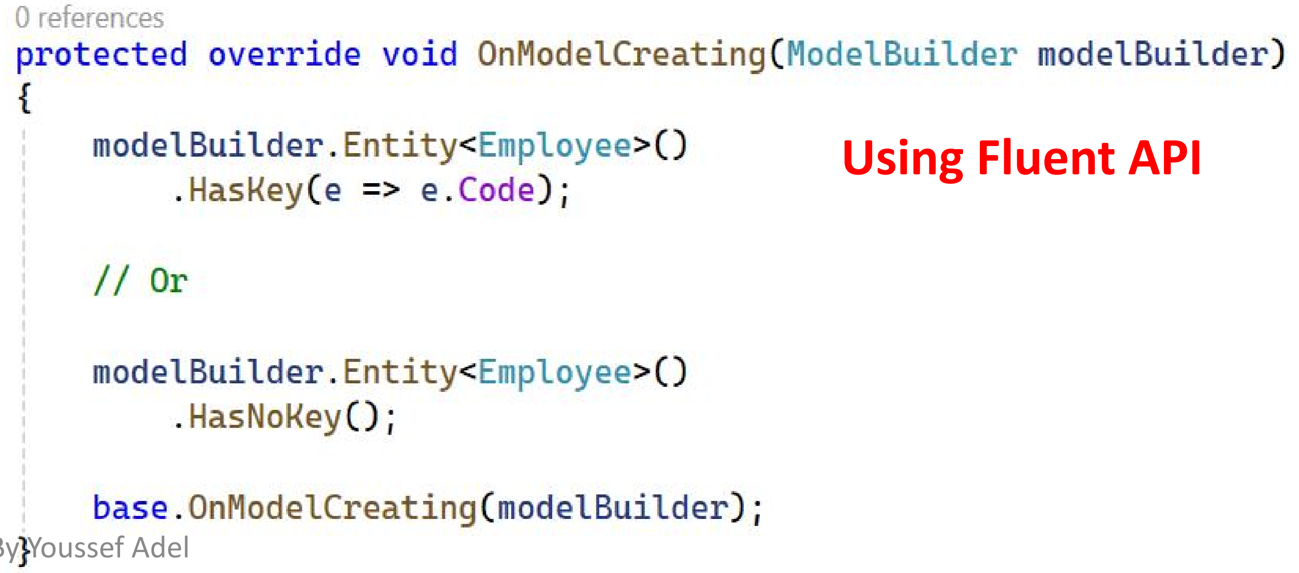
Data Annotation VS Fluent API

- Set primary key
- If the property named like [Id, EmployeeId] it will be PK by default.
- If you have another name you must tell efcore that it will be the PK
- You have another option to mark the table with HasNoKey()



Using Data Annotations

```
3 references
class Employee
{
    [Key]
    1 reference
    public int Code { get; set; }

    0 references
    public string Name { get; set; } = null!;

    0 references
    public string? Department { get; set; }

    0 references
    public double Salary { get; set; }
}
```

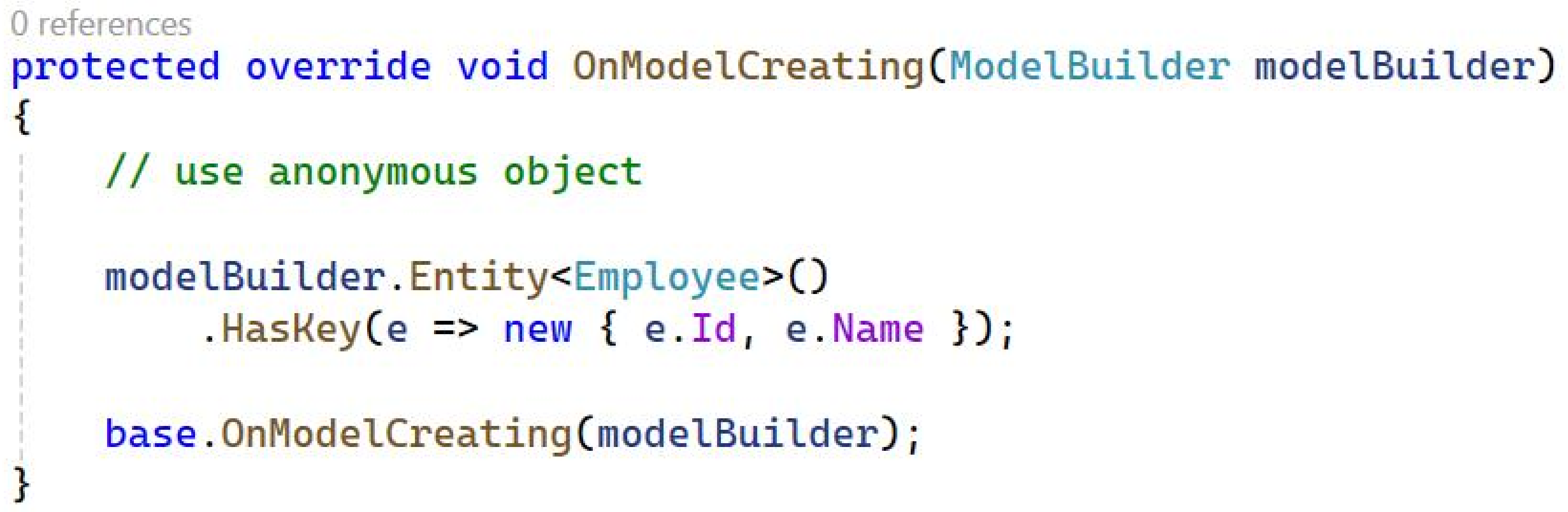


Using Fluent API

```
0 references
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Employee>()
        .HasKey(e => e.Code);

    // Or

    modelBuilder.Entity<Employee>()
        .HasNoKey();

    base.OnModelCreating(modelBuilder);
}
```

Data Annotation VS Fluent API

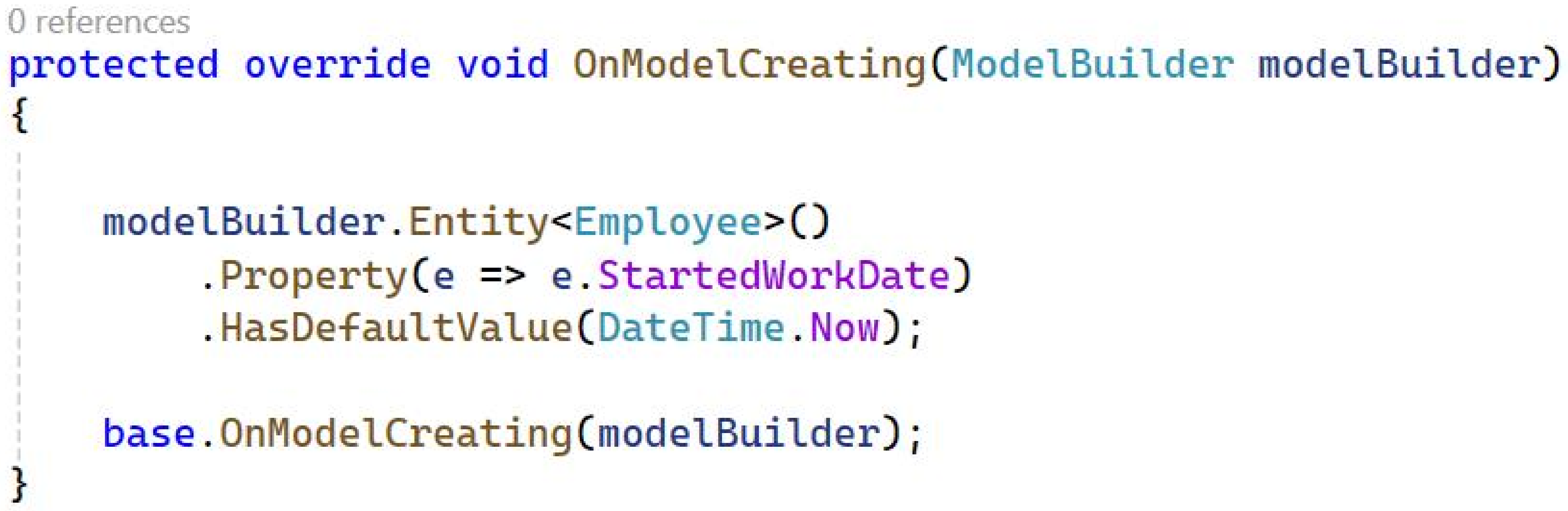
- Set composite key
- It is the PK that made of two or more columns.
- These columns must be unique toghether.
- It can be done only using Fluent API.



```
0 references
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    // use anonymous object

    modelBuilder.Entity<Employee>()
        .HasKey(e => new { e.Id, e.Name });

    base.OnModelCreating(modelBuilder);
}
```

Data Annotation VS Fluent API

- Set default value
- It can be done only using Fluent API.

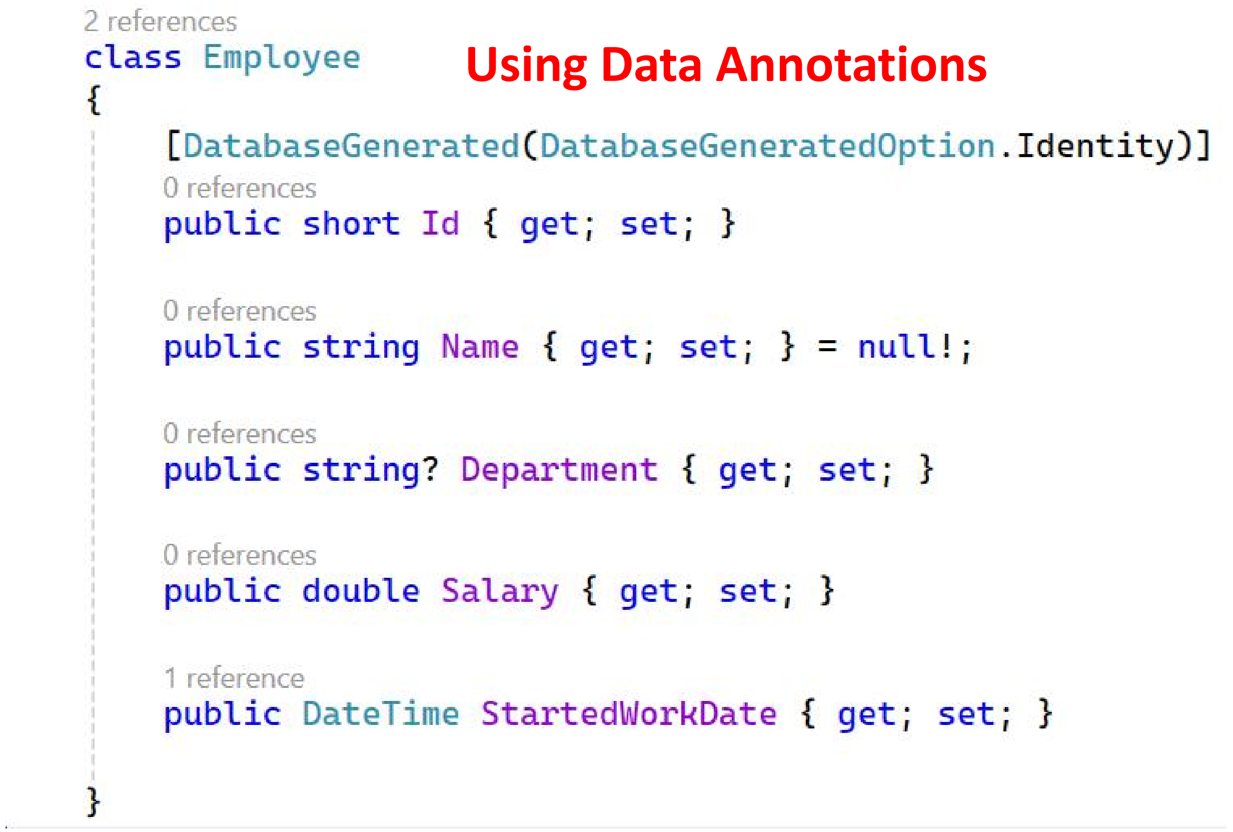


```
0 references
protected override void OnModelCreating(ModelBuilder modelBuilder)
{

    modelBuilder.Entity<Employee>()
        .Property(e => e.StartedWorkDate)
        .HasDefaultValue(DateTime.Now);

    base.OnModelCreating(modelBuilder);
}
```

Data Annotation VS Fluent API

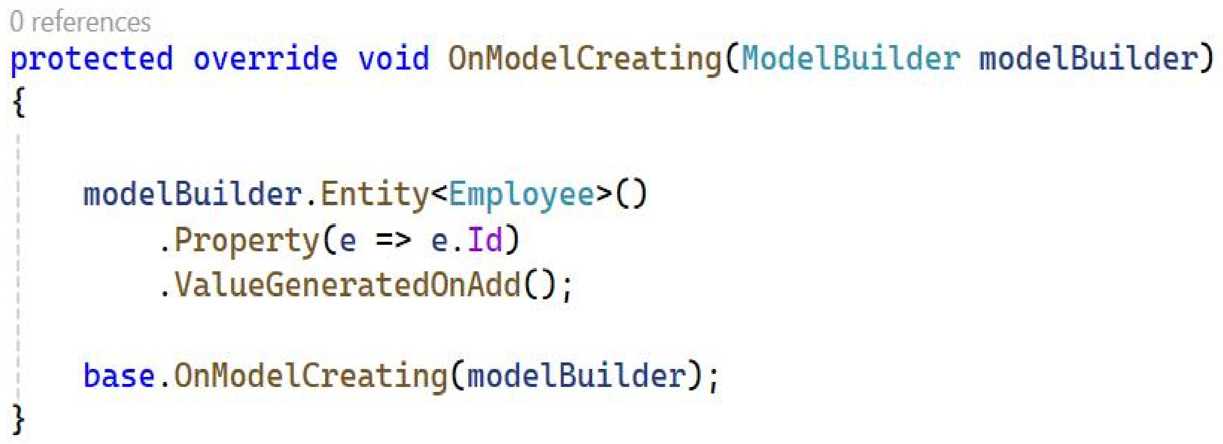
- Set Identity to the column

Using Data Annotations



```
2 references
class Employee
{
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    0 references
    public short Id { get; set; }

    0 references
    public string Name { get; set; } = null!;

    0 references
    public string? Department { get; set; }

    0 references
    public double Salary { get; set; }

    1 reference
    public DateTime StartedWorkDate { get; set; }

}
```

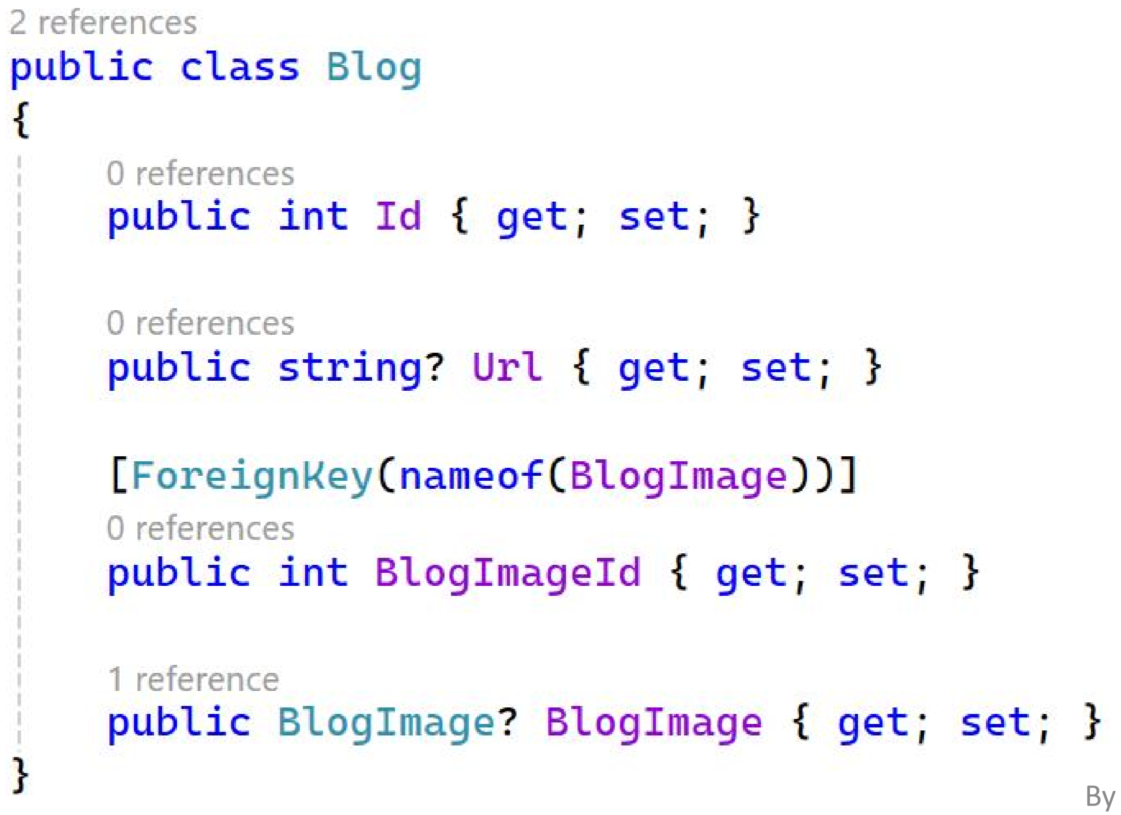
Using Fluent API



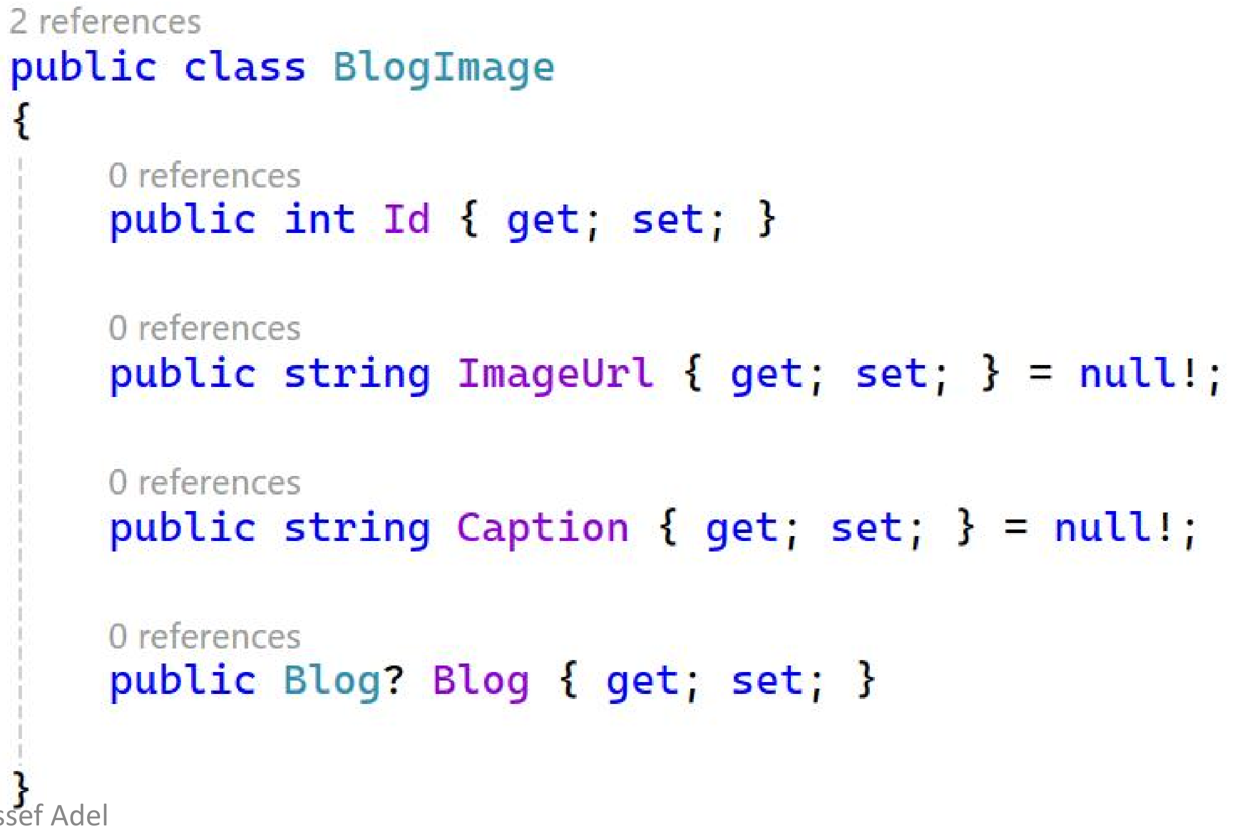
```
0 references
protected override void OnModelCreating(ModelBuilder modelBuilder)
{

    modelBuilder.Entity<Employee>()
        .Property(e => e.Id)
        .ValueGeneratedOnAdd();

    base.OnModelCreating(modelBuilder);
}
```

Relationships

- One to one relationship



```
2 references
public class Blog
{
    0 references
    public int Id { get; set; }

    0 references
    public string? Url { get; set; }

    [ForeignKey(nameof(BlogImage))]
    0 references
    public int BlogImageId { get; set; }

    1 reference
    public BlogImage? BlogImage { get; set; }
}
```

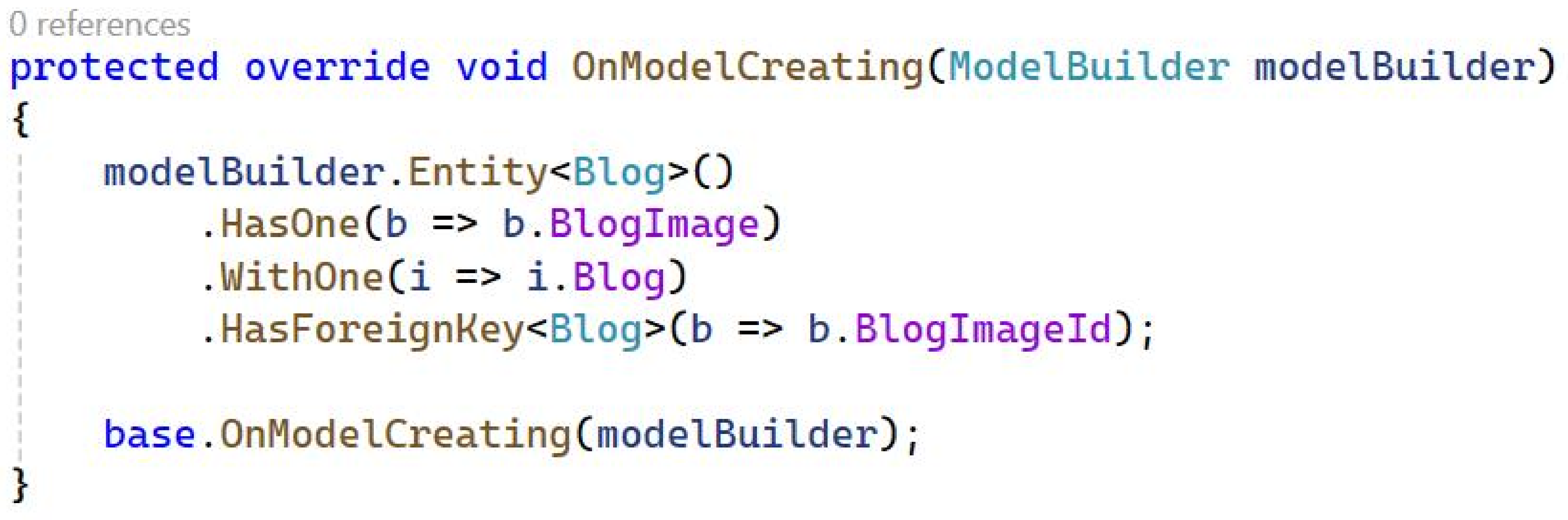


```
2 references
public class BlogImage
{
    0 references
    public int Id { get; set; }

    0 references
    public string ImageUrl { get; set; } = null!;

    0 references
    public string Caption { get; set; } = null!;

    0 references
    public Blog? Blog { get; set; }
}
```

```
                     _/\__
               ---==/    \\
         ___  ___   |.    \|\
        | __|| __|  |  )   \\\
        | _| | _|   \_/ |  //|\\
        |___||_|       /   \\\/\\
```

Relationships

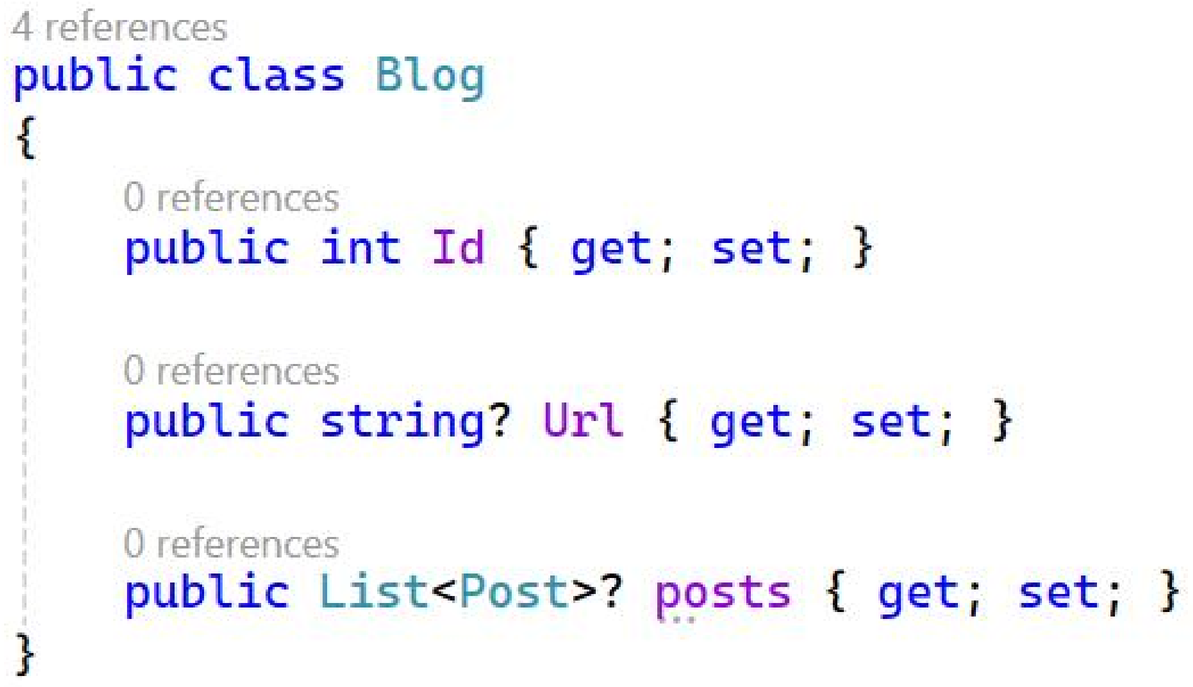
- Configure one to one relationship using Fluent API.



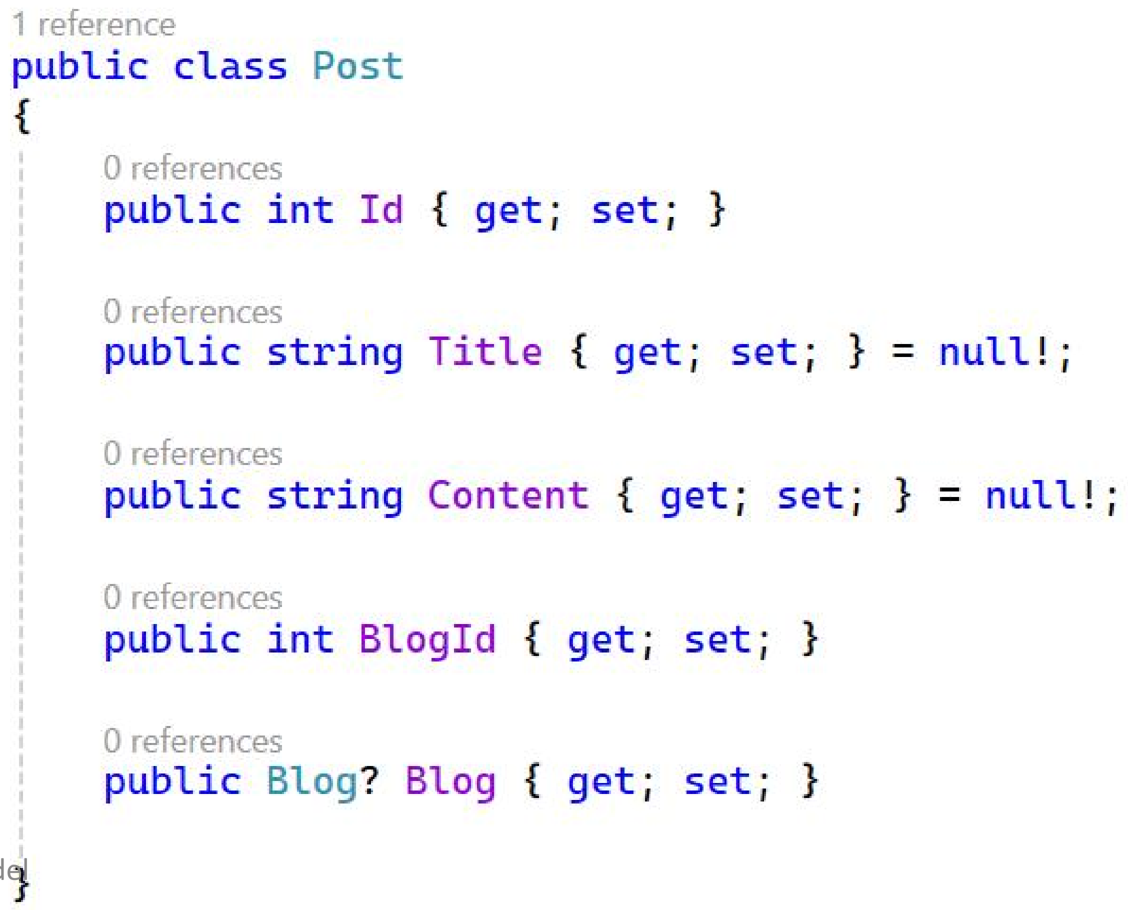
```
0 references
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Blog>()
        .HasOne(b => b.BlogImage)
        .WithOne(i => i.Blog)
        .HasForeignKey<Blog>(b => b.BlogImageId);

    base.OnModelCreating(modelBuilder);
}
```

Relationships

- One to many relationship



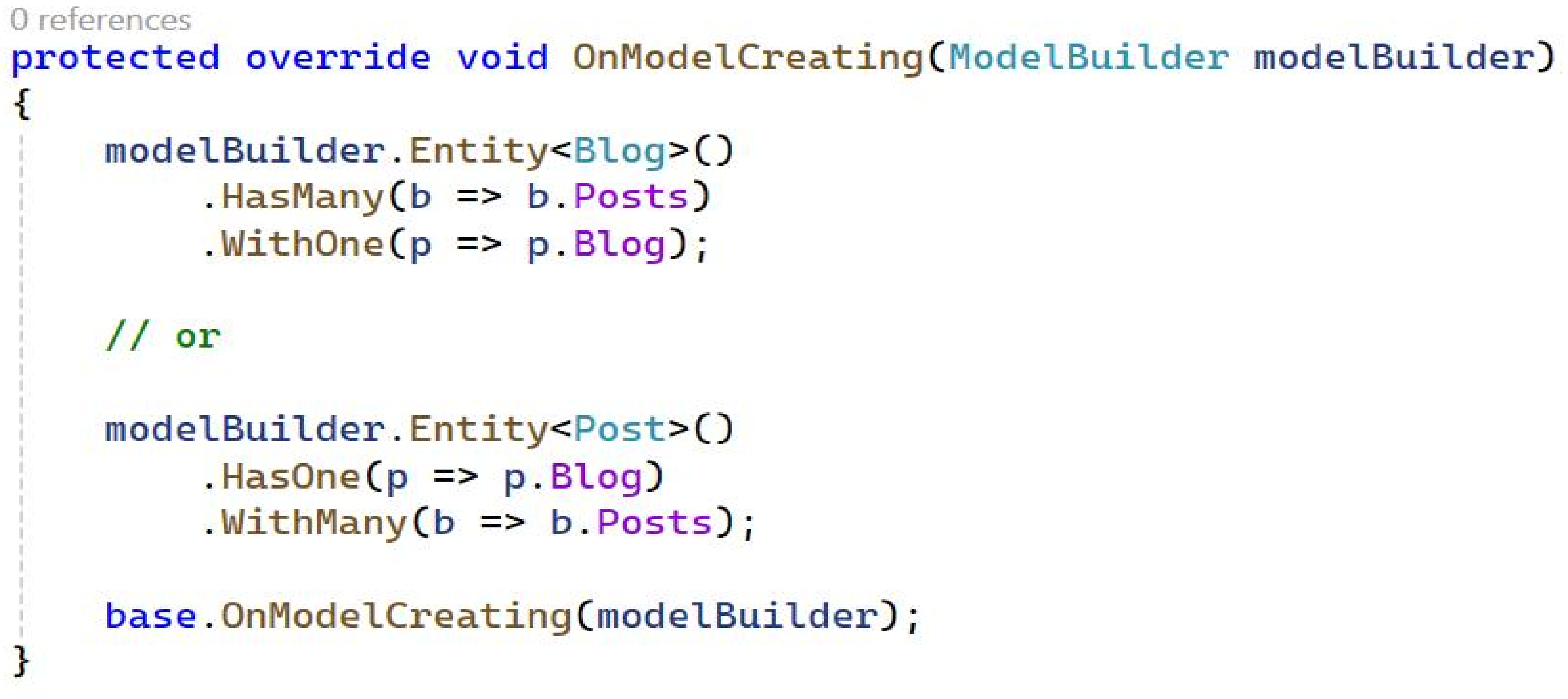
```
4 references
public class Blog
{
    0 references
    public int Id { get; set; }

    0 references
    public string? Url { get; set; }

    0 references
    public List<Post>? posts { get; set; }
}
```



```
1 reference
public class Post
{
    0 references
    public int Id { get; set; }

    0 references
    public string Title { get; set; } = null!;

    0 references
    public string Content { get; set; } = null!;

    0 references
    public int BlogId { get; set; }

    0 references
    public Blog? Blog { get; set; }
}
```

Relationships

- Configure one to many relationship using Fluent API.



```
0 references
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Blog>()
        .HasMany(b => b.Posts)
        .WithOne(p => p.Blog);

    // or

    modelBuilder.Entity<Post>()
        .HasOne(p => p.Blog)
        .WithMany(b => b.Posts);

    base.OnModelCreating(modelBuilder);
}
```

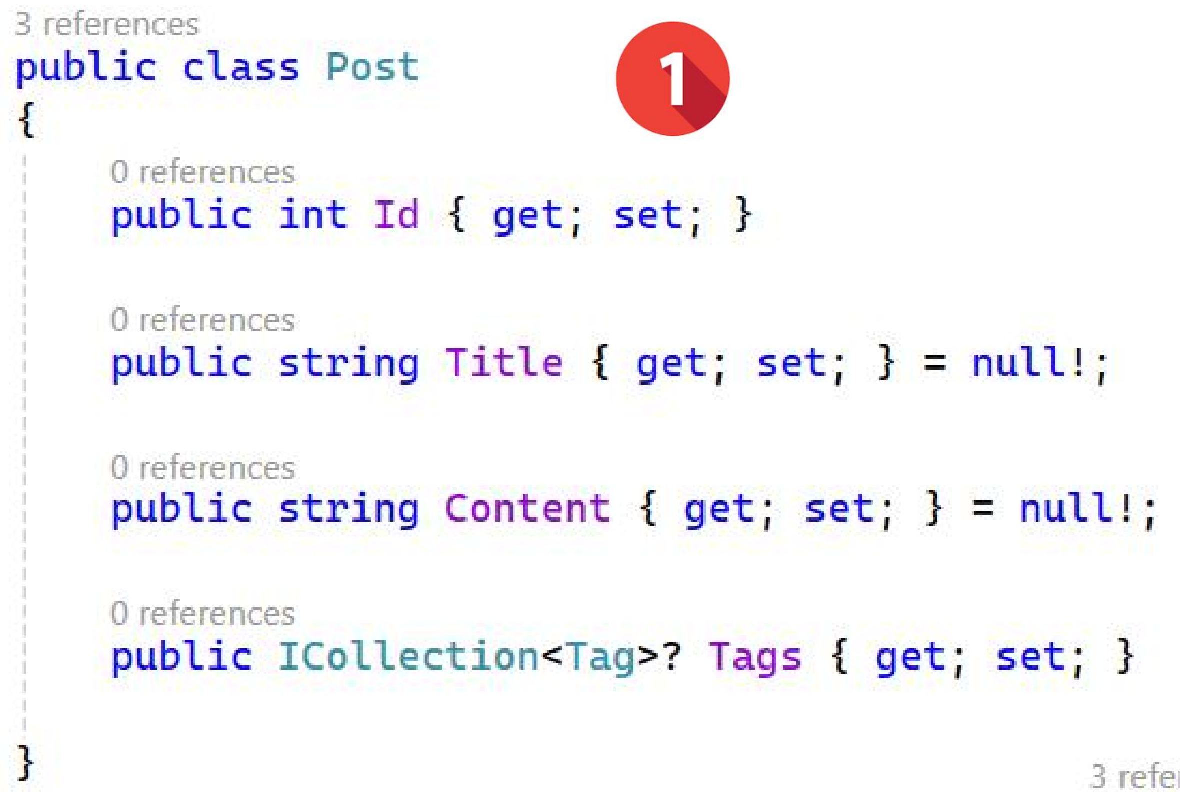
```
             _/\__
       ---==/   \\
 ___  ___   |.    \|\
| __|| __|  |  )   \\\
| _| | _|   \_/ |  //|\\
|___||_|       /   \\\/\\
```

Relationships

- Many to many relationship.
- Here we need the third table to set the m to m relationship.
- There are many ways to configure this relationship.
- We will take the just two simple ways to achive MtoM.

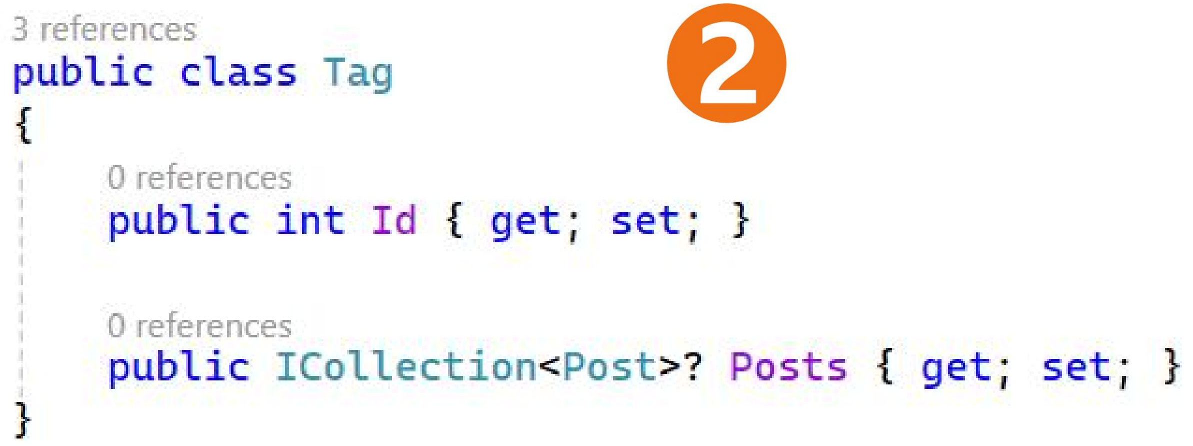
1st way : Just using

```
                     _/\__
               ---==/    \\
         ___  ___   |.    \|\
        | __|| __|  |  )   \\\
        | _| | _|   \_/ |  //|\\
        |___||_|       /   \\\/\\
```



1

```
3 references
public class Post
{
    0 references
    public int Id { get; set; }

    0 references
    public string Title { get; set; } = null!;

    0 references
    public string Content { get; set; } = null!;

    0 references
    public ICollection<Tag>? Tags { get; set; }

}
```



2

```
3 references
public class Tag
{
    0 references
    public int Id { get; set; }

    0 references
    public ICollection<Post>? Posts { get; set; }
}
```

By Youssef Adel

2nd way : Explicit table

1

```
2 references
public class Post
{
    0 references
    public int Id { get; set; }

    0 references
    public string Title { get; set; } = null!;

    0 references
    public string Content { get; set; } = null!;

    0 references
    public ICollection<PostTag> Tags { get; set; } = new List<PostTag>();

}
```

2

```
2 references
public class Tag
{
    0 references
    public int Id { get; set; }

    0 references
    public ICollection<PostTag> Posts { get; set; } = new List<PostTag>();
}
```

3

```
6 references
public class PostTag
{
    1 reference
    public int TagId { get; set; }

    0 references
    public Tag? Tag { get; set; }

    1 reference
    public int PostId { get; set; }

    0 references
    public Post? Post { get; set; }

}
```

By Youssef Adel

EFCore Data Query

- Select all records:

```
var stores = _context.Stores.ToList();
```

- Find by Id:

```
var customer = _context.Customers.Find(100);
```

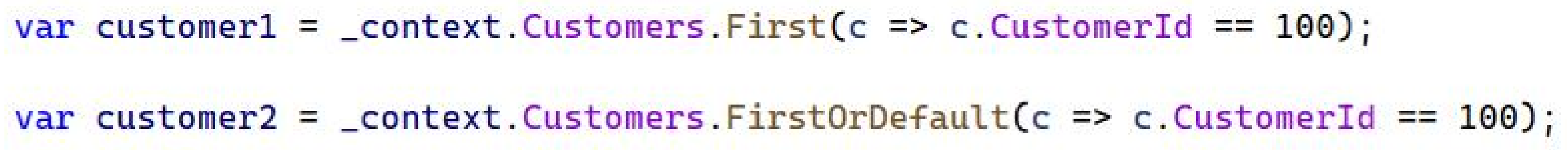
- Select one item using single:

```
var customer1 = _context.Customers.Single(c => c.CustomerId == 100);

var customer2 = _context.Customers.SingleOrDefault(c => c.CustomerId == 100);
```

```
                     _/\__
               ---==/    \\
         ___  ___   |.    \|\
        | __|| __|  |  )   \\\
        | _| | _|   \_/ |  //|\\
        |___||_|       /   \\\/\\
```

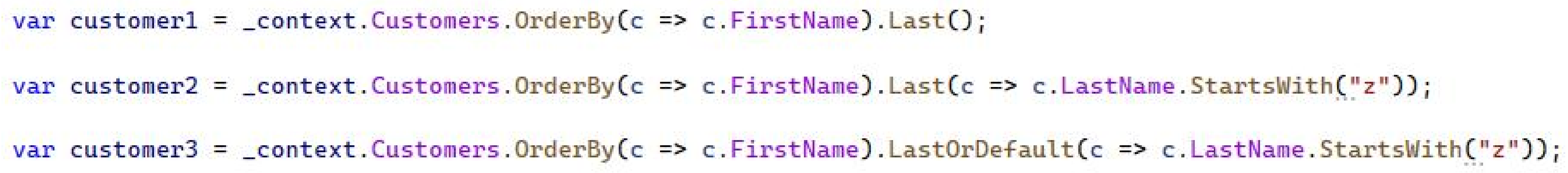
EFCore Data Query

- Select first that meets a condition.



```
var customer1 = _context.Customers.First(c => c.CustomerId == 100);

var customer2 = _context.Customers.FirstOrDefault(c => c.CustomerId == 100);
```

- Select last that meets a condition, must have a sort order.



```
var customer1 = _context.Customers.OrderBy(c => c.FirstName).Last();

var customer2 = _context.Customers.OrderBy(c => c.FirstName).Last(c => c.LastName.StartsWith("z"));

var customer3 = _context.Customers.OrderBy(c => c.FirstName).LastOrDefault(c => c.LastName.StartsWith("z"));
```

EFCore Data Query

- Filtering data using Where()
- Where() make the filtering in server side and return on the data after filtering.

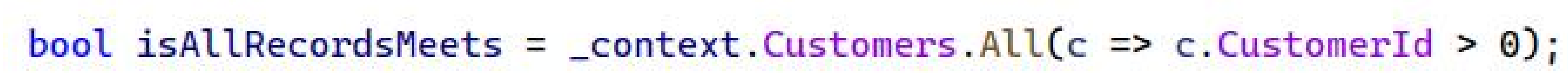
```
var customers = _context.Customers.Where(c => c.CustomerId > 500).ToList();
```

- Find if any record in the table, or any record meets the condition.

```
bool isThereAnyRecords = _context.Customers.Any();

bool isThereAnyRecordsMeets = _context.Customers.Any(c => c.LastName.Equals("Hamada"));
```

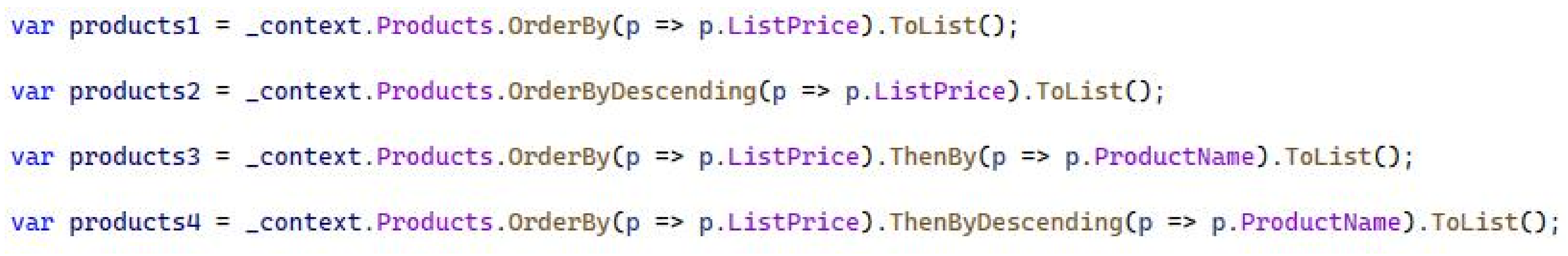
EFCore Data Query

- Find if all records meets the condition.



```
bool isAllRecordsMeets = _context.Customers.All(c => c.CustomerId > 0);
```

- Sorting data

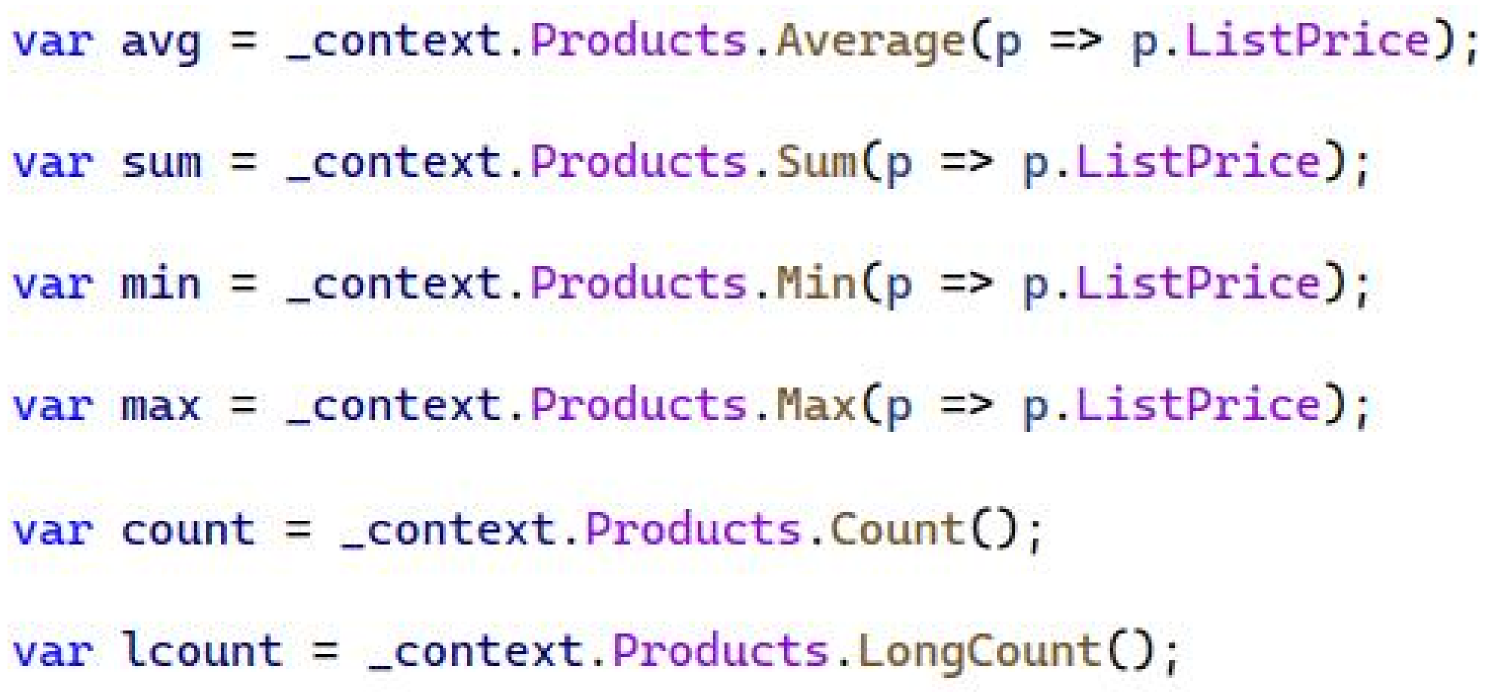


```
var products1 = _context.Products.OrderBy(p => p.ListPrice).ToList();

var products2 = _context.Products.OrderByDescending(p => p.ListPrice).ToList();

var products3 = _context.Products.OrderBy(p => p.ListPrice).ThenBy(p => p.ProductName).ToList();

var products4 = _context.Products.OrderBy(p => p.ListPrice).ThenByDescending(p => p.ProductName).ToList();
```

```
                     _/\__
               ---==/    \\
         ___  ___   |.    \|\
        | __|| __|  |  )   \\\
        | _| | _|   \_/ |  //|\\
        |___||_|       /   \\\/\\
```

EFCore Data Query

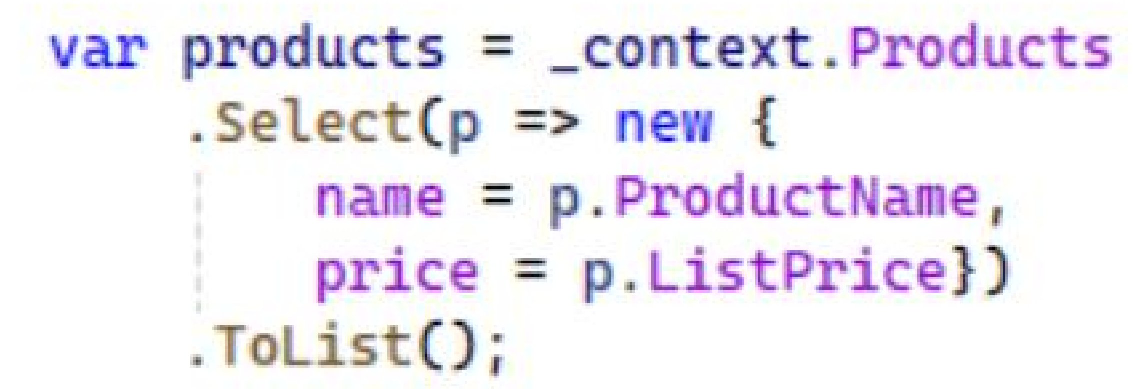
- Aggregate functions



```
var avg = _context.Products.Average(p => p.ListPrice);

var sum = _context.Products.Sum(p => p.ListPrice);

var min = _context.Products.Min(p => p.ListPrice);

var max = _context.Products.Max(p => p.ListPrice);

var count = _context.Products.Count();

var lcount = _context.Products.LongCount();
```

EFCore Data Query

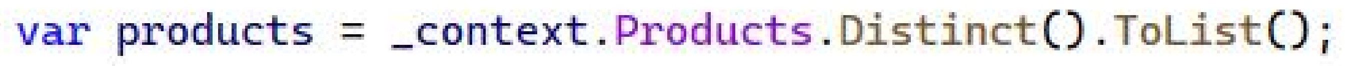
- Data projection with Select()
- Means to change data shape when selecting.
- Very Important !!!



```
var products = _context.Products
    .Select(p => new {
        name = p.ProductName,
        price = p.ListPrice})
    .ToList();
```

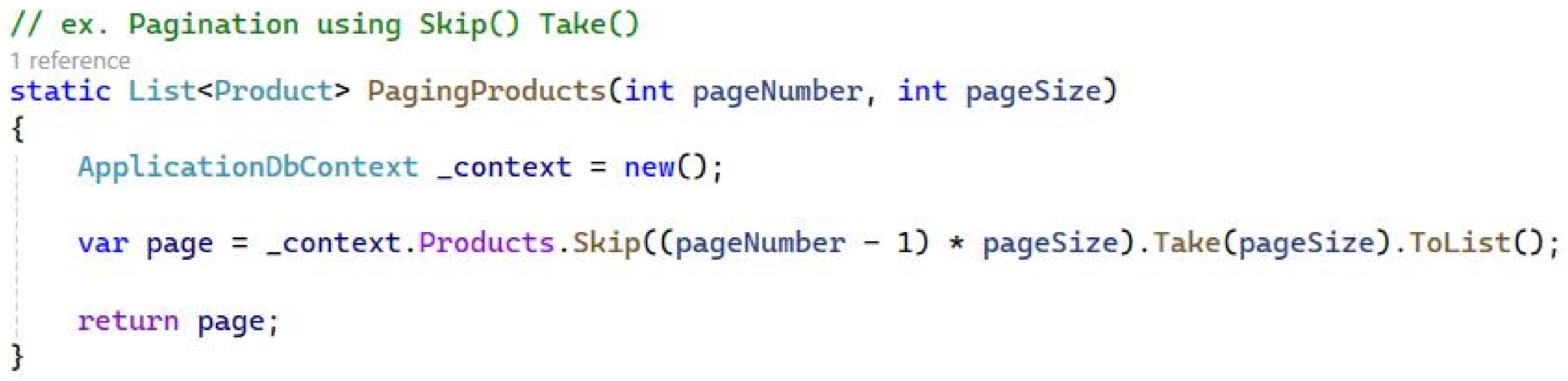
EFCore Data Query

- Select distinct values:



```
var products = _context.Products.Distinct().ToList();
```

- Skip() and Take()

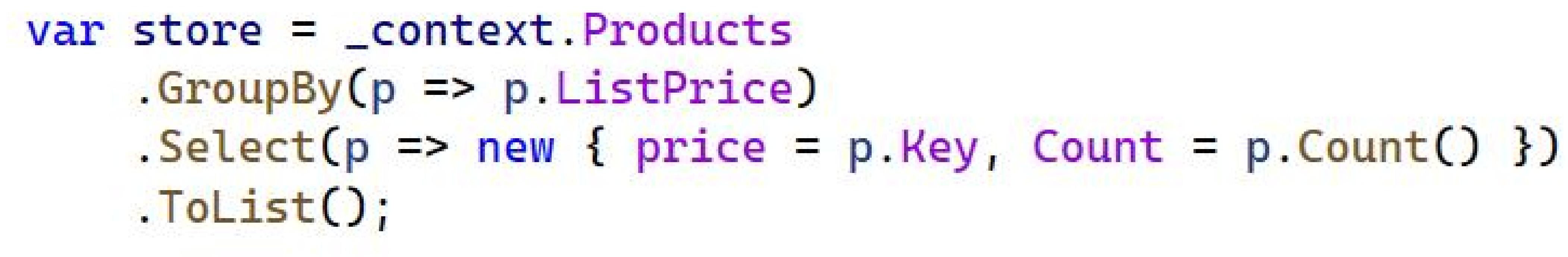


```
// ex. Pagination using Skip() Take()
1 reference
static List<Product> PagingProducts(int pageNumber, int pageSize)
{
    ApplicationDbContext _context = new();

    var page = _context.Products.Skip((pageNumber - 1) * pageSize).Take(pageSize).ToList();

    return page;
}
```

```
                     _/\__
               ---==/    \\
         ___  ___   |.    \|\
        | __|| __|  |  )   \\\
        | _| | _|   \_/ |  //|\\
        |___||_|       /   \\\/\\
```

EFCore Data Query

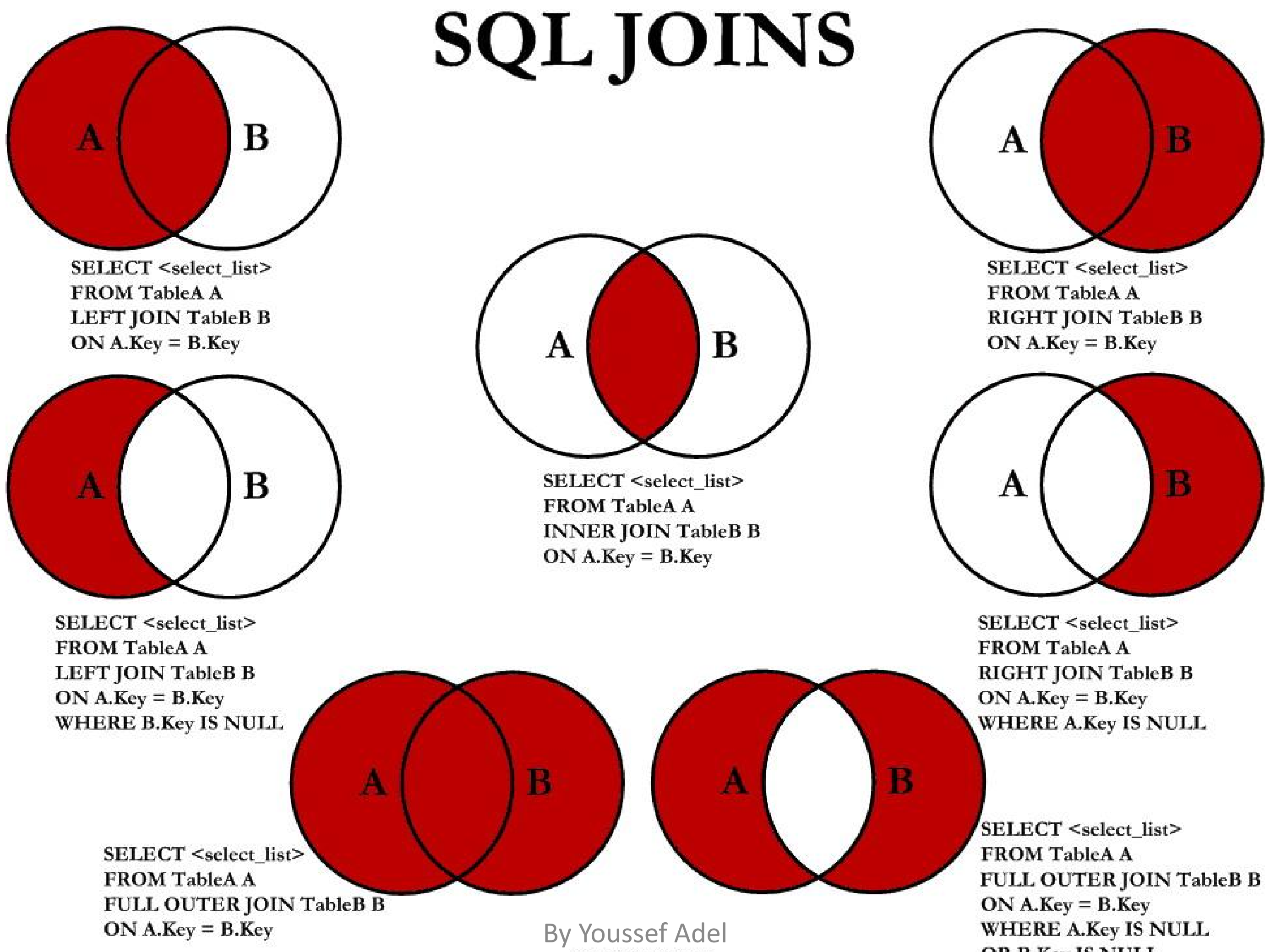
- GroupeBy()



```
var store = _context.Products
    .GroupBy(p => p.ListPrice)
    .Select(p => new { price = p.Key, Count = p.Count() })
    .ToList();
```

```
                     _/\__
               ---==/    \\
         ___  ___   |.    \|\
        | __|| __|  |  )   \\\
        | _| | _|   \_/ |  //|\\
        |___||_|       /   \\\/\\
```

EFCore Data Query



```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```

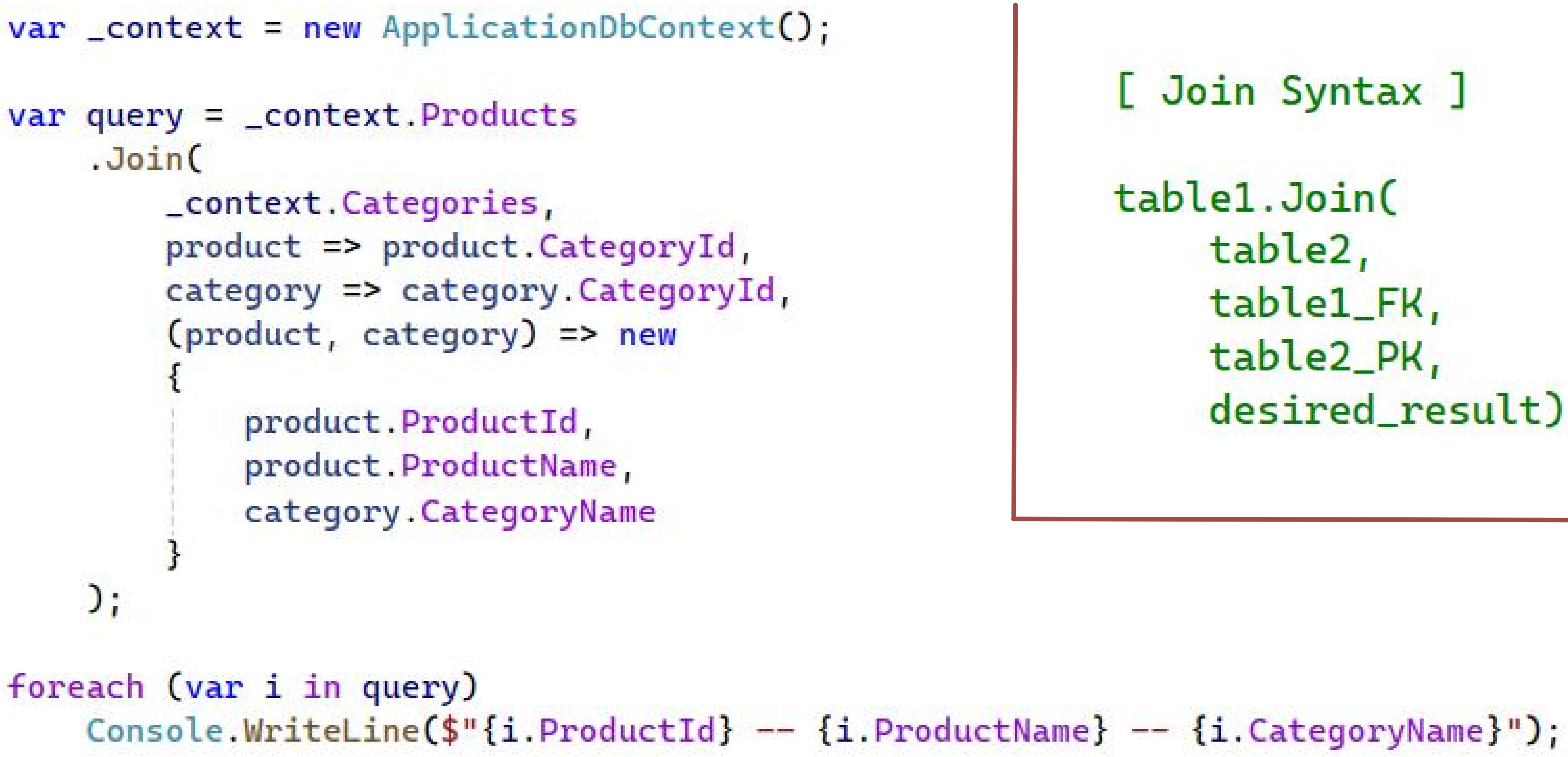
```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```

```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```



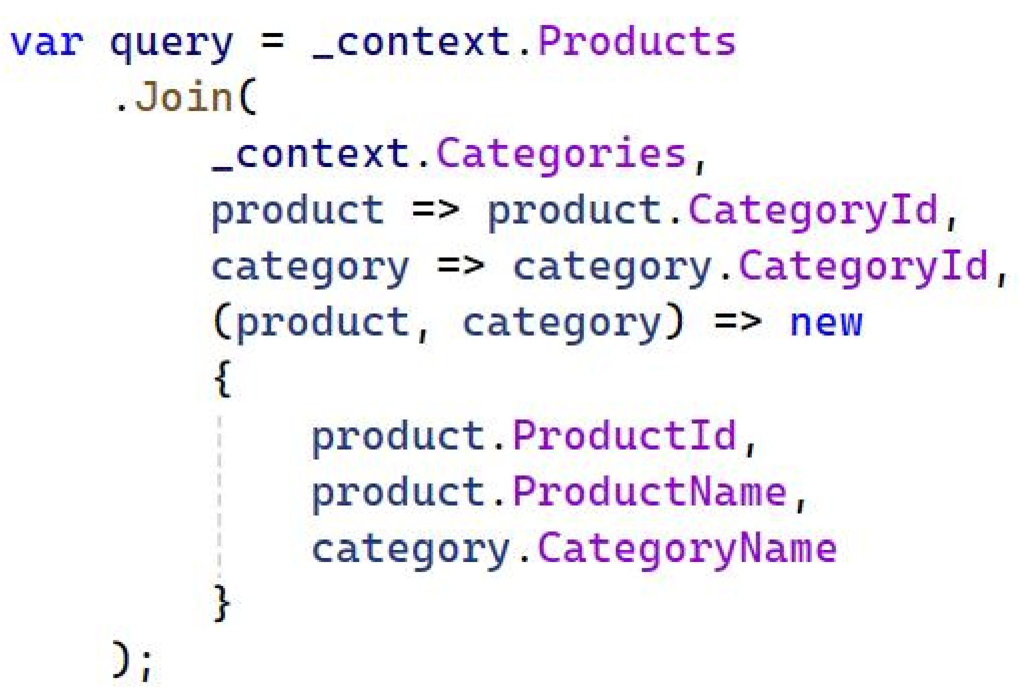
```csharp
var _context = new ApplicationDbContext();

var query = _context.Products
    .Join(
        _context.Categories,
        product => product.CategoryId,
        category => category.CategoryId,
        (product, category) => new
        {
            product.ProductId,
            product.ProductName,
            category.CategoryName
        }
    );

foreach (var i in query)
    Console.WriteLine($"{i.ProductId} -- {i.ProductName} -- {i.CategoryName}");
```

```
[ Join Syntax ]

table1.Join(
    table2,
    table1_FK,
    table2_PK,
    desired_result)
```

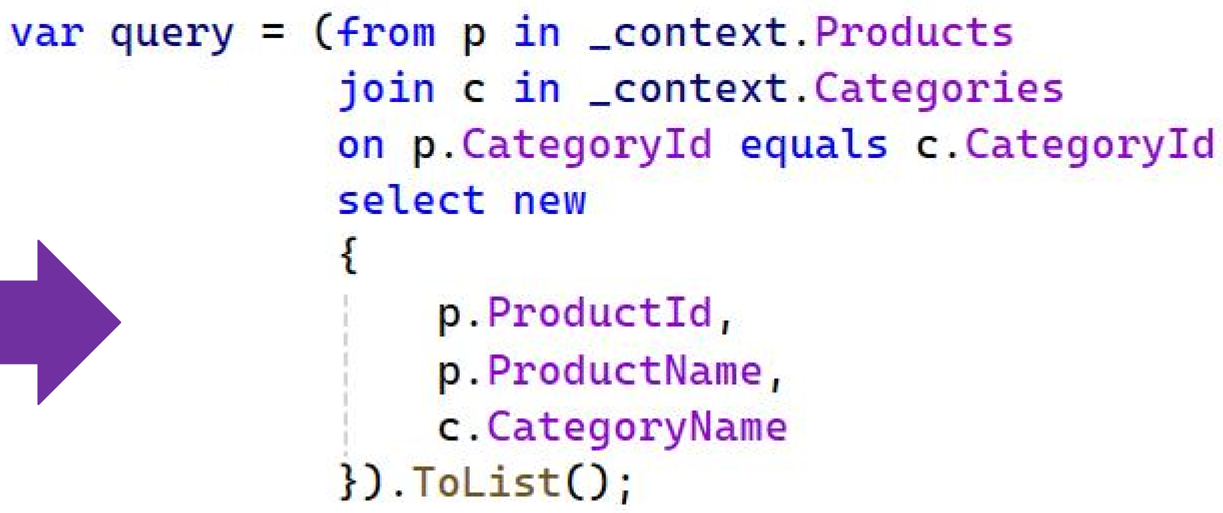
EFCore Data Query

- using System.Linq;

Search How??
Make yourself curious....



```
var query = _context.Products
    .Join(
        _context.Categories,
        product => product.CategoryId,
        category => category.CategoryId,
        (product, category) => new
        {
            product.ProductId,
            product.ProductName,
            category.CategoryName
        }
    );
```



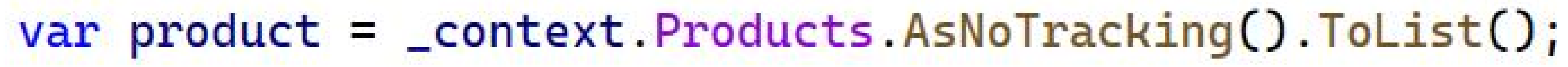
```
var query = (from p in _context.Products
             join c in _context.Categories
             on p.CategoryId equals c.CategoryId
             select new
             {
                 p.ProductId,
                 p.ProductName,
                 c.CategoryName
             }).ToList();
```

Tracking in EFCore

- Tracking means to track or record the cahnges you make on the database and send them toghether when you call _context.SaveChanges()

- EFCore default behavior is Tracking, but if you query data that you don’t need to make changes on?

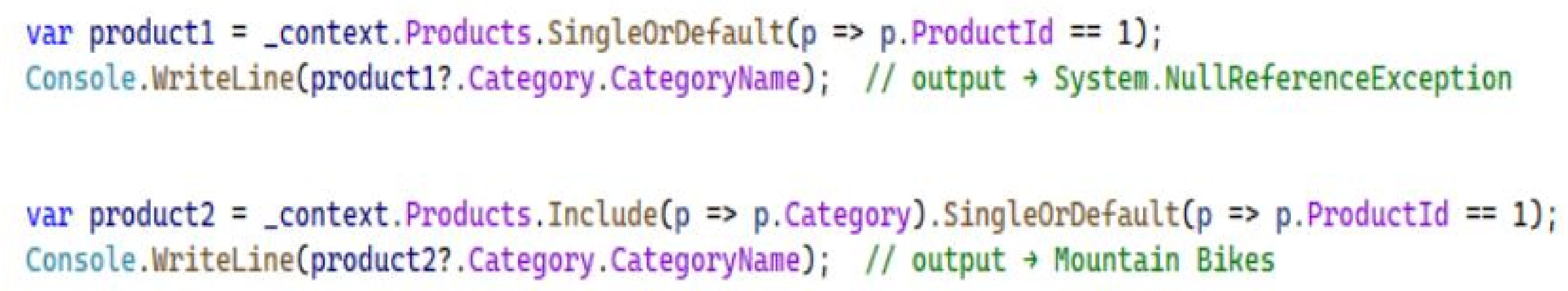
- We use .AsNoTracking() in this condition.



```
var product = _context.Products.AsNoTracking().ToList();
```

Loading in EFCore

- Eager Loading:
  - means to load the table and all specified related tables.
  - You can include more using .ThenInclude()
  - Eager loading is heavy for application performance.



```
var product1 = _context.Products.SingleOrDefault(p => p.ProductId == 1);
Console.WriteLine(product1?.Category.CategoryName);  // output → System.NullReferenceException


var product2 = _context.Products.Include(p => p.Category).SingleOrDefault(p => p.ProductId == 1);
Console.WriteLine(product2?.Category.CategoryName);  // output → Mountain Bikes
```

Search for Explixit Loading

```
                     _/\__
               ---==/    \\
         ___  ___   |.    \|\
        | __|| __|  |  )   \\\
        | _| | _|   \_/ |  //|\\
        |___||_|       /   \\\/\\
```

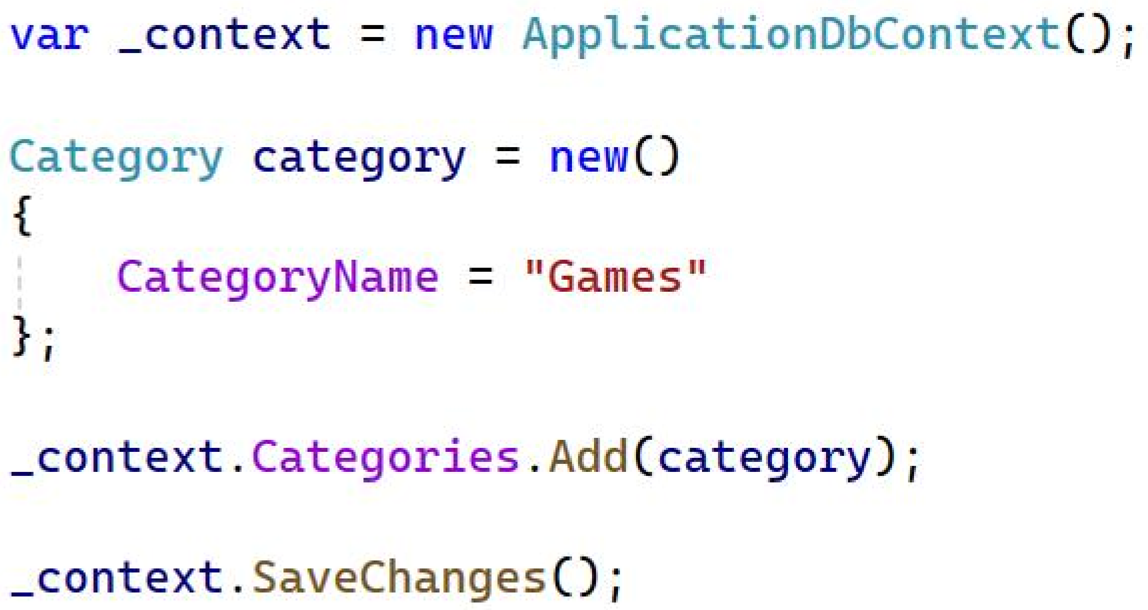
Loading in EFCore

- Lazy Loading:
  - means to load the table and load other related tables only when it is used.
  - How to use lazy loading??

1. Download package Microsoft.EntityFrameworkCore.Proxies
2.
```
optionsBuilder.UseLazyLoadingProxies().UseSqlServer(connstring);
```
3.
```
// will load [ Products ] only.
var product = _context.Products.SingleOrDefault(p => p.ProductId == 1);

// Here [ Categories ] will be loaded when it is used.
Console.WriteLine(product?.Category.CategoryName);
```

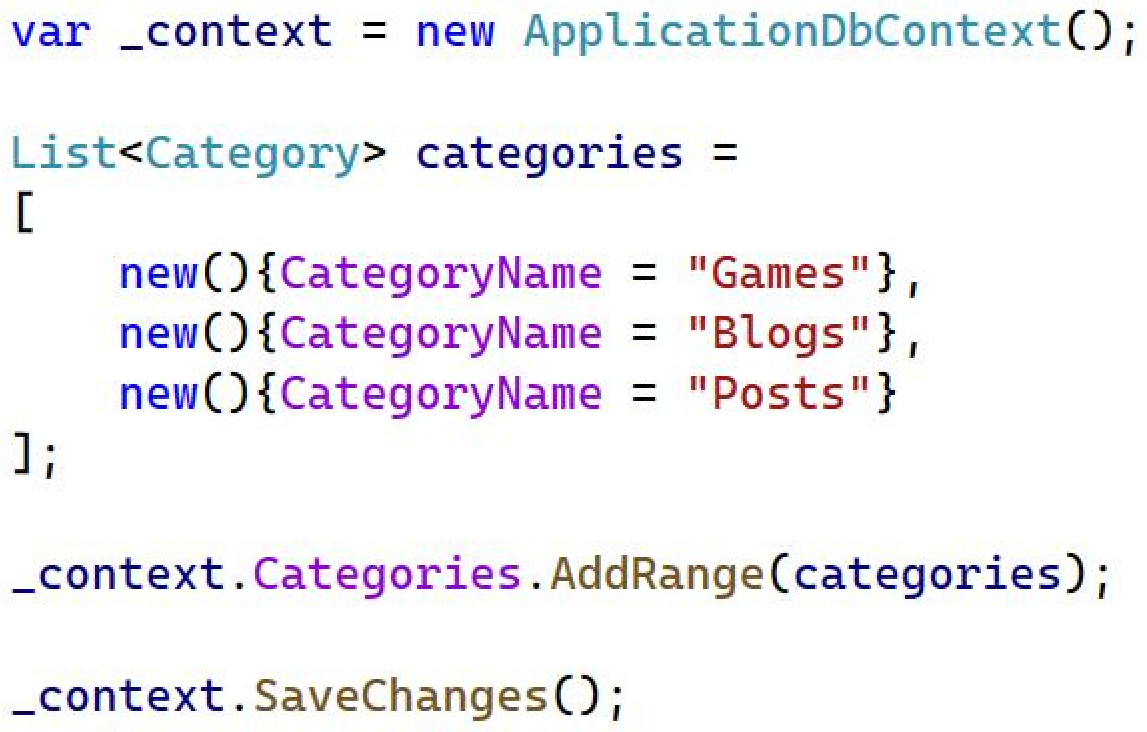
CRUD Opertations In EFCore

- Add()



```
var _context = new ApplicationDbContext();

Category category = new()
{
    CategoryName = "Games"
};

_context.Categories.Add(category);

_context.SaveChanges();
```

AddRange()

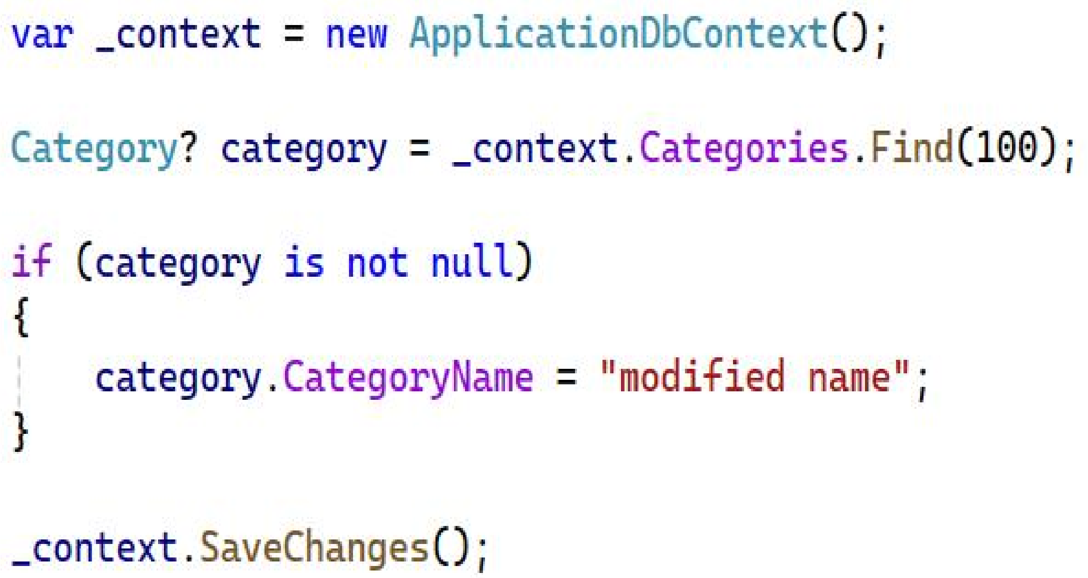


```
var _context = new ApplicationDbContext();

List<Category> categories =
[
    new(){CategoryName = "Games"},
    new(){CategoryName = "Blogs"},
    new(){CategoryName = "Posts"}
];

_context.Categories.AddRange(categories);

_context.SaveChanges();
```

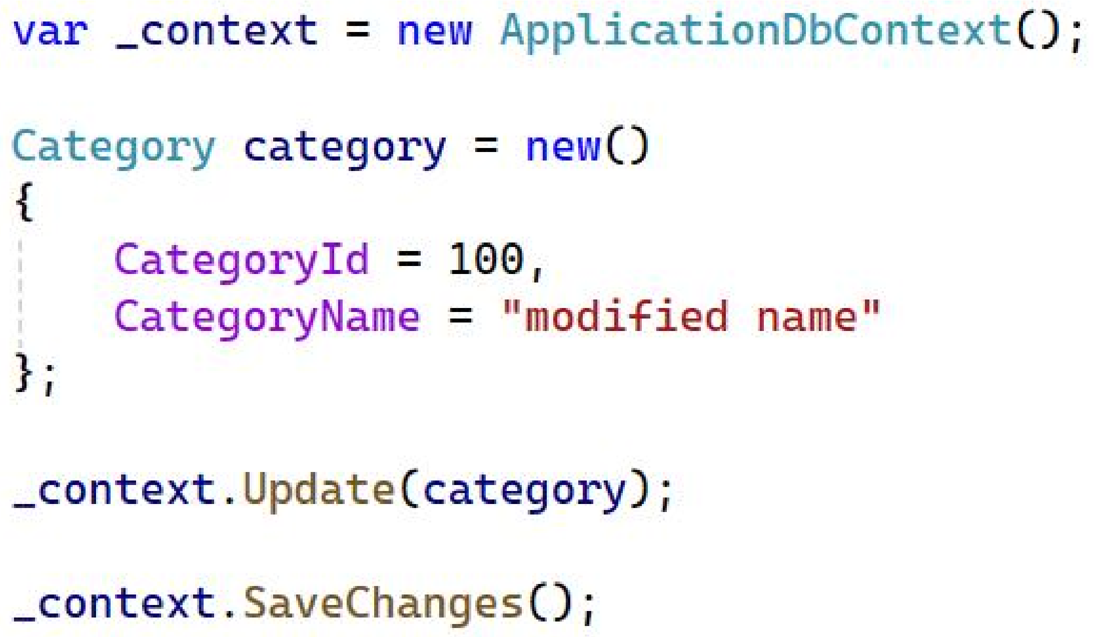
By Youssef Adel

CRUD Opertations In EFCore

- Updating in two ways:



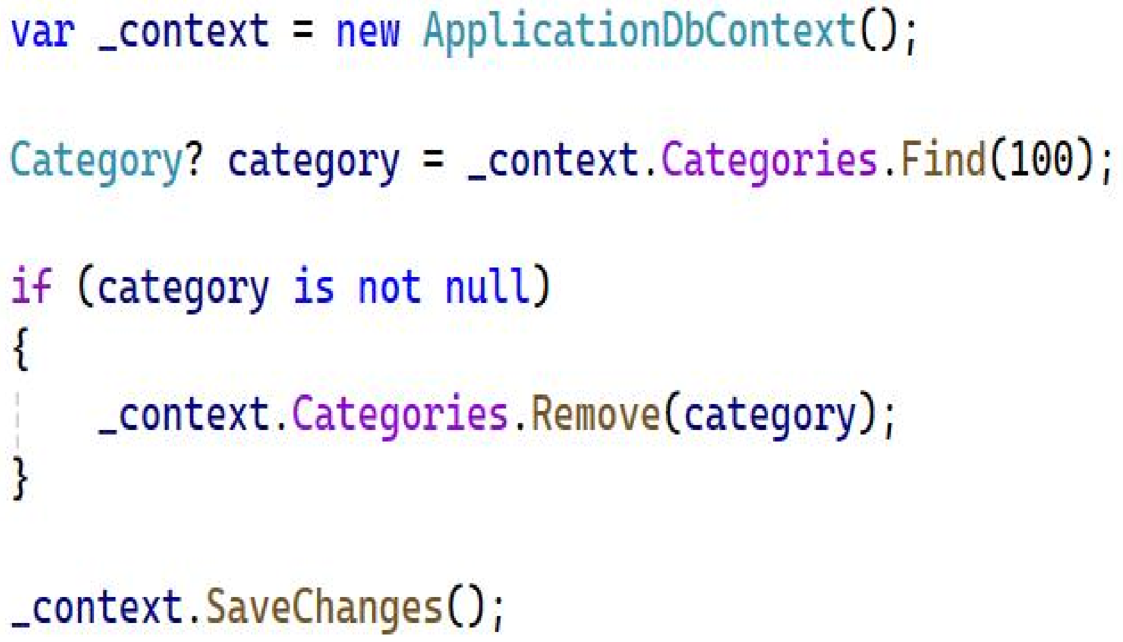
```
var _context = new ApplicationDbContext();

Category? category = _context.Categories.Find(100);

if (category is not null)
{
    category.CategoryName = "modified name";
}

_context.SaveChanges();
```



```
var _context = new ApplicationDbContext();

Category category = new()
{
    CategoryId = 100,
    CategoryName = "modified name"
};

_context.Update(category);

_context.SaveChanges();
```

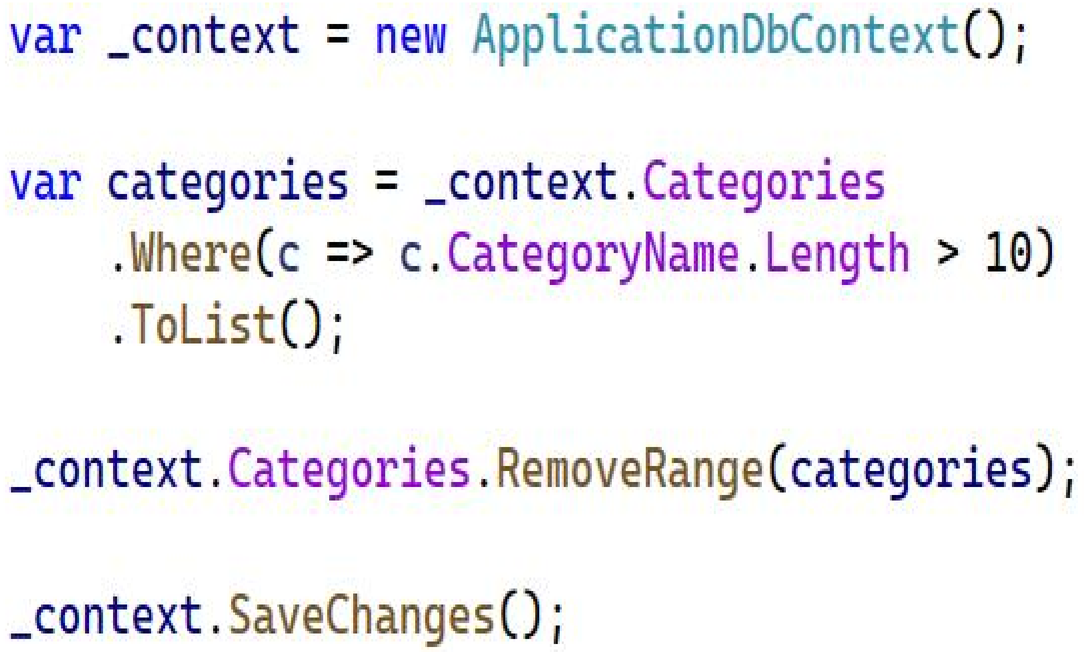
CRUD Opertations In EFCore

- Remove()



```csharp
var _context = new ApplicationDbContext();

Category? category = _context.Categories.Find(100);

if (category is not null)
{
    _context.Categories.Remove(category);
}

_context.SaveChanges();
```

RemoveRange()



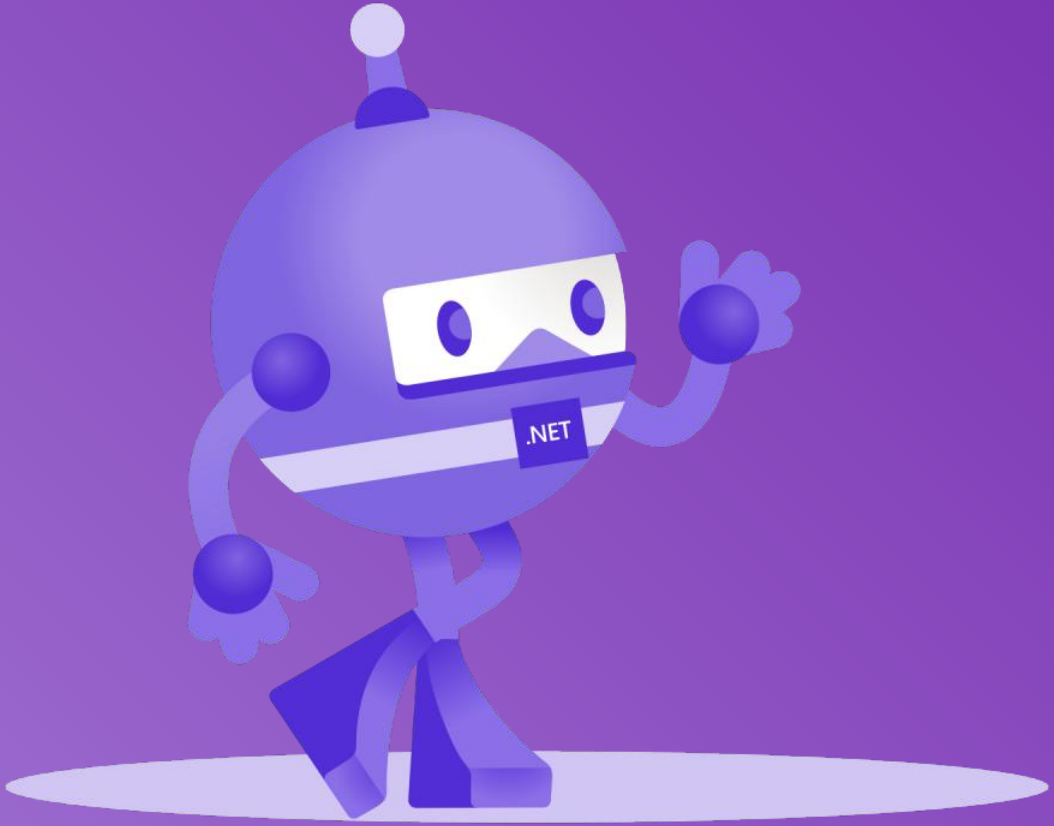
```csharp
var _context = new ApplicationDbContext();

var categories = _context.Categories
    .Where(c => c.CategoryName.Length > 10)
    .ToList();

_context.Categories.RemoveRange(categories);

_context.SaveChanges();
```

Note: Deleting related data is done according to your deleting behavior (cascade, restrict, ...) so make sure you have the desired delete action.

```
Console.WriteLine("Thank You!");
```

By Youssef Adel

Goodbye, GDSC Family



It's been a pleasure meeting all of you, and I wish you success in the years ahead. Remember, "it's not difficult; it's just new to you"

By Youssef Adel