VIA University College

DNP-S18

Data Access

How to persist and access data

Introducing Entity Framework Core

What is Entity Framework and why do we need an ORM?

Using Entity Framework Core

How to persist and access data using EF Core

The Repository Pattern

Abstracting Data Access

LINQ

How to use Language Integrated Query to query the database

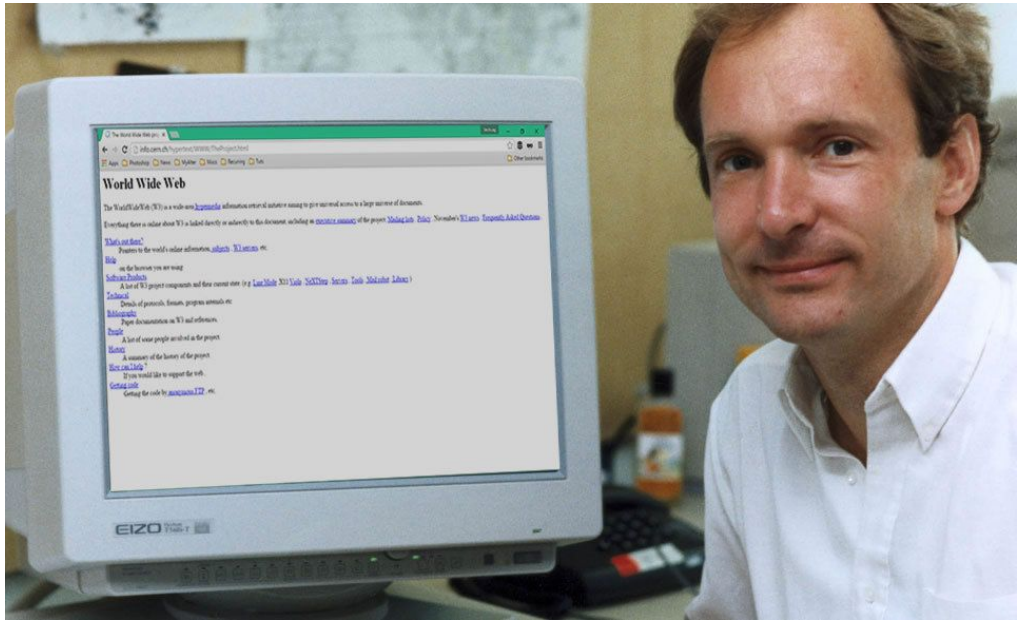
Exercises

Create and manage your own database!

A Word From Tim Berners-Lee

"Data is a precious thing and will last longer than the systems themselves."

- Tim Berners-Lee



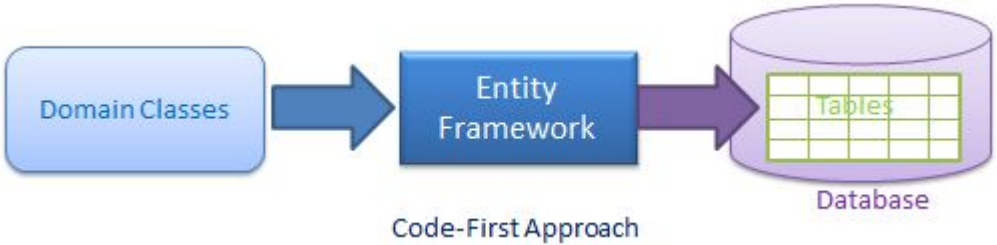
Traditional Relational DB Access


```
// Create a connection
using (SqlConnection conn = new SqlConnection(Configuration["ConnectionStrings:DefaultConnection"]))
{
    conn.Open();

    // Create a command to join Customer and CustomerContactInfo tables
    SqlCommand command = conn.CreateCommand();
    command.CommandText = @"
        SELECT cust.FirstName, cust.LastName, contact.EmailAddress
        FROM [dbo].[Customer] AS cust
        JOIN [dbo].[CustomerContactInfo] AS contact
        ON cust.CustomerID = contact.CustomerID
        WHERE cust.MiddleName IS NULL";

    // Execute the command and obtain a data reader.
    using (var reader = command.ExecuteReader())
    {
        while(reader.Read())
        {
            // Write a response using values from the reader.
            Console.WriteLine($@"
                {reader["FirstName"]}
                {reader["LastName"]})
                {reader["EmailAdress"]}");
        }
    }
}
```

Entity Framework Core

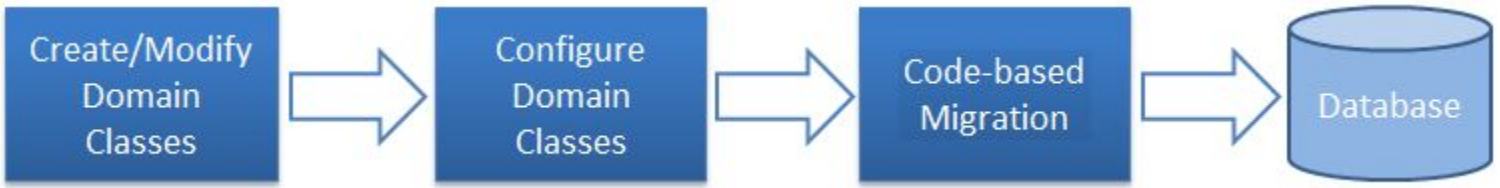
Entity Framework (EF) Core is an object-relational mapper (ORM) that enables persisting domain-specific objects to and from a data source.

- Eliminates the need for most of the data access code developers would typically need to write
- Build on top of ADO.NET
- “Code First” approach
    - Focus on the domain of your application and start creating classes for your domain (vs. design your database first and then create the classes which match your database design)



Entity Framework Core Docs

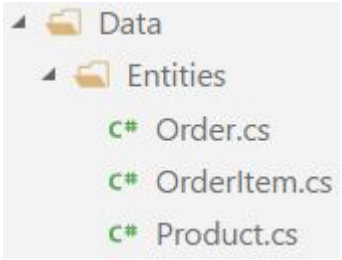
EF Core Work Flow



DB Access With EF Core


```
using (ApplicationDbContext context = new ApplicationDbContext())
{
    var query = from c in context.Customer
                where c.MiddleName == null
                select new {
                    FirstName = c.FirstName,
                    LastName = c.LastName,
                    EmailAdress = c.EmailAddress
                };

    foreach (var c in query)
    {
        Console.WriteLine($@"
            {c.FirstName}
            {c.LastName}
            {c.EmailAdress}");
    }
}
```

Creating Entities




```
public class OrderItem
{
    public int Id { get; set; }
    public Product Product { get; set; }
    public int Quantity { get; set; }
    public decimal UnitPrice { get; set; }
    public Order Order { get; set; }
}
```

By convention, a property named **Id** or **<type name>Id** will be configured as the key of an entity.

```
public class Order
{
    public int Id { get; set; }
    public DateTime OrderDate { get; set; }
    public string OrderNumber { get; set; }
    public ICollection<OrderItem> Items { get; set; }
}
```

One-to-many relationship

Table columns

...

Key

```
public class Product
{
    public int Id { get; set; }
    public string Category { get; set; }
    public string Size { get; set; }
    public decimal Price { get; set; }
    public string Title { get; set; }
    public string ArtDescription { get; set; }
    public string ArtDating { get; set; }
    public string ArtId { get; set; }
    public string Artist { get; set; }
    public DateTime ArtistBirthDate { get; set; }
    public DateTime ArtistDeathDate { get; set; }
    public string ArtistNationality { get; set; }
}
```

Configuring Domain Classes


```
public class Movie
{
    public int ID { get; set; }

    [StringLength(60, MinimumLength = 3)]
    [Required]
    public string Title { get; set; }

    [Display(Name = "Release Date")]
    [DataType(DataType.Date)]
    public DateTime ReleaseDate { get; set; }

    [Range(1, 100)]
    [DataType(DataType.Currency)]
    public decimal Price { get; set; }

    [RegularExpression(@"^[A-Z]+[a-zA-Z""'\s-]*$")]
    [Required]
    [StringLength(30)]
    public string Genre { get; set; }

    [RegularExpression(@"^[A-Z]+[a-zA-Z""'\s-]*$")]
    [StringLength(5)]
    [Required]
    public string Rating { get; set; }
}
```

Data annotations

Namespace
System.ComponentModel.DataAnnotations

Configuring Domain Classes

Using annotations

```
public class Blog
{
    public int BlogId { get; set; }
    [Required]
    public string Url { get; set; }
}
```

Using Fluent API

```
class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Blog>()
            .Property(b => b.Url)
            .IsRequired();
    }
}
```

DbContext

- To work with EF Core, you need a subclass of DbContext.
- DbContext is a bridge between your domain and the database. It holds properties representing collections of the entities your application will work with.
- It is the primary class that is responsible for interacting with the database.

```
public class ApplicationDbContext : DbContext
{
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options)
    {
    }

    public DbSet<Product> Products { get; set; }
    public DbSet<Order> Orders { get; set; }
}
```

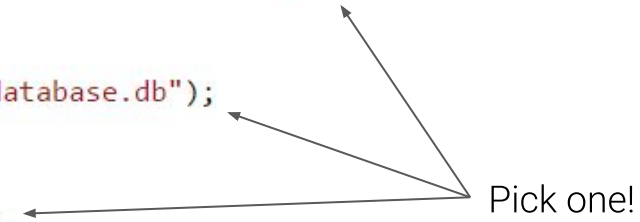
Tables

Why no DbSet for OrderItems?
Only create DbSets for enitities you want to query directly!

Configuring a DbContext

In Startup.cs

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<ApplicationDbContext>(options =>
    {
        // SQL Server
        options.UseSqlServer(Configuration["ConnectionStrings:DefaultConnection"]);

        // SQLite
        options.UseSqlite("Data Source = database.db");

        // In Memory Database
        options.UseInMemoryDatabase("db");
    });

    services.AddMvc();
}
```



Configuring a DbContext

Configuration


```
public class Startup
{
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    public IConfiguration Configuration { get; }
```

Available through Dependency Injection!

Example usage

```
Configuration.GetConnectionString("DefaultConnection");

Configuration.GetSection("SomeSection").Value;

Configuration["section:subsection:suboption"];
```

appsettings.json

appsettings.json

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=localhost\\SQLEXPRESS;Database=dnp;Trusted_Connection=True;MultipleActiveResultSets=true;"
  },
  "Logging": {
    "IncludeScopes": false,
    "Debug": {
      "LogLevel": {
        "Default": "Warning"
      }
    },
    "Console": {
      "LogLevel": {
        "Default": "Warning"
      }
    }
  }
}
```

Database Providers

Microsoft SQL Server (MSSQL)
Fully featured relational database with great tooling, performance and security.

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

SQLite
Light-weight and self-contained. It's a code library without any other dependencies. There's nothing to configure.

There's no database server. The client and the server run in the same process.

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

In Memory Database
For testing only.

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

Database providers

List of DB providers with sample code

EF CLI Tooling

Your project can use the Entity Framework CLI commands

```
dotnet ef
```

Commands

database	Commands to manage the database.
dbcontext	Commands to manage DbContext types.
migrations	Commands to manage migrations.

EF Core CLI Tools

Migrations

Migrations provide a way to incrementally apply schema changes to the database to keep it in sync with your EF Core model while preserving existing data in the database.

```
dotnet ef migrations add [migration name]
```

Looks at the datastores of our app.
Adds code that can go from empty database to our actual database.

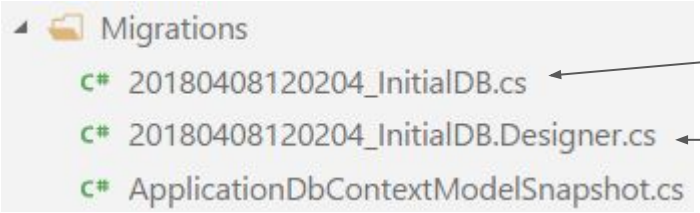
```
dotnet ef database update
```

Applies the migration to the database to create the schema.

Migrations

Migrations

All automatically generated!
No need to hand-edit the files.



The main migrations file. Contains the operations necessary to apply (and revert) the migration.

The migrations metadata file. Contains information used by EF.

A snapshot of the current model. Used to determine what changed when adding the next migration.

Empty DB → DB that can store our information

Using DbContext

You can request an instance of your DbContext type in any service that needs it.

E.g. inject your DbContext into the constructor of your Web API controller

```
[Route("api")]
public class ValuesController : Controller
{
    private readonly ApplicationDbContext _context;

    public ValuesController(ApplicationDbContext context)
    {
        _context = context;
    }
```

... The rest is history! (See Web API slides)

Seeding the Database


```
public static class DbInitializer
{
    public static void Initialize(ApplicationDbContext context)
    {
        context.Database.EnsureCreated();

        if (context.Products.Any())
        {
            return; // DB has been seeded
        }

        var products = new Product[]
        {
            new Product { Category = "Candy", Price = 3.5M, Title = "Candy Art" },
            new Product { Category = "Cats", Price = 1337, Title = "Computer Cats" },
            new Product { Category = "Programming", Price = 42, Title = "DNP is art" }
        };

        foreach(Product p in products)
        {
            context.Add(p);
        }
        context.SaveChanges();
    }
}
```

Data Seeding

Data Seeding in EF Core

Seeding the Database

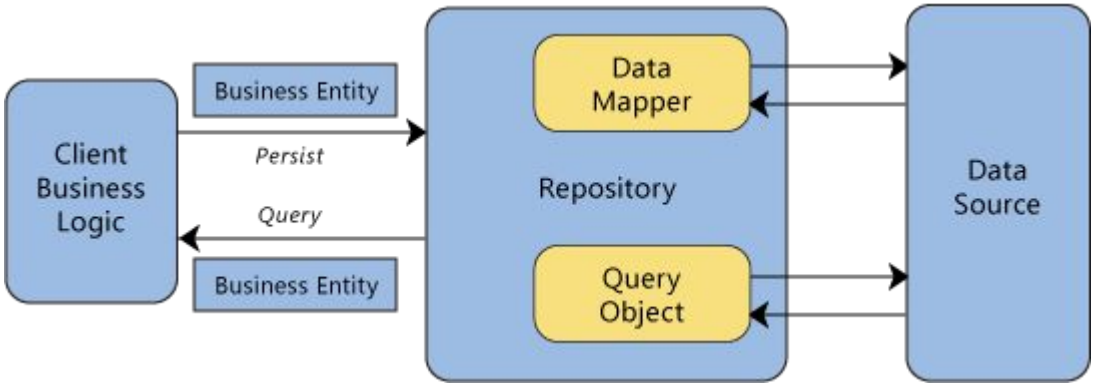
Update Main method to take advantage of the seeder!

```
public static void Main(string[] args)
{
    var host = BuildWebHost(args);

    using (var scope = host.Services.CreateScope())
    {
        var services = scope.ServiceProvider;
        try
        {
            var context = services.GetRequiredService<ApplicationDbContext>();
            DbInitializer.Initialize(context);
        }
        catch (Exception ex)
        {
            var logger = services.GetRequiredService<ILogger<Program>>();
            logger.LogError(ex, "An error occurred while seeding the database.");
        }
    }

    host.Run();
}
```

Entity Framework Core Demo

The Repository Pattern

- The repository pattern is a pattern for abstracting data access
- The job of the repository is to expose the different calls to the database
- EF Core can be considered an implementation of the repository pattern



Using the Repository Pattern

Why Use The Repository Pattern

The Repository Pattern, Example


```
public class DbRepository
{
    private readonly ApplicationDbContext _context;

    public DbRepository(ApplicationDbContext context)
    {
        _context = context;
    }

    public IEnumerable<Product> GetAllProducts()
    {
        return _context.Products
            .OrderBy(p => p.Title)
            .ToList();
    }

    public IEnumerable<Product> GetProductsByCategory(string category)
    {
        return _context.Products
            .Where(p => p.Category == category)
            .ToList();
    }

    public bool SaveAll()
    {
        return _context.SaveChanges() > 0;
    }
}
```

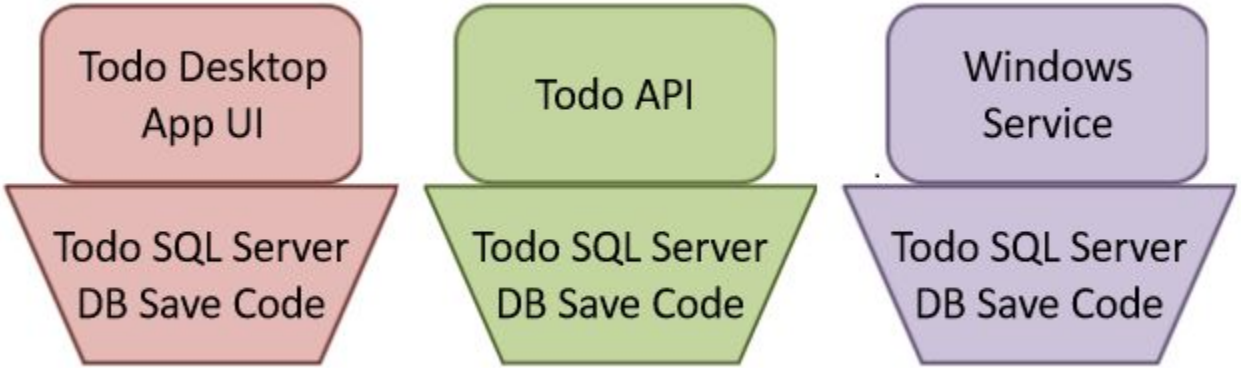
In ConfigureServices (Startup.cs)

```
services.AddSingleton<DbRepository>();
```

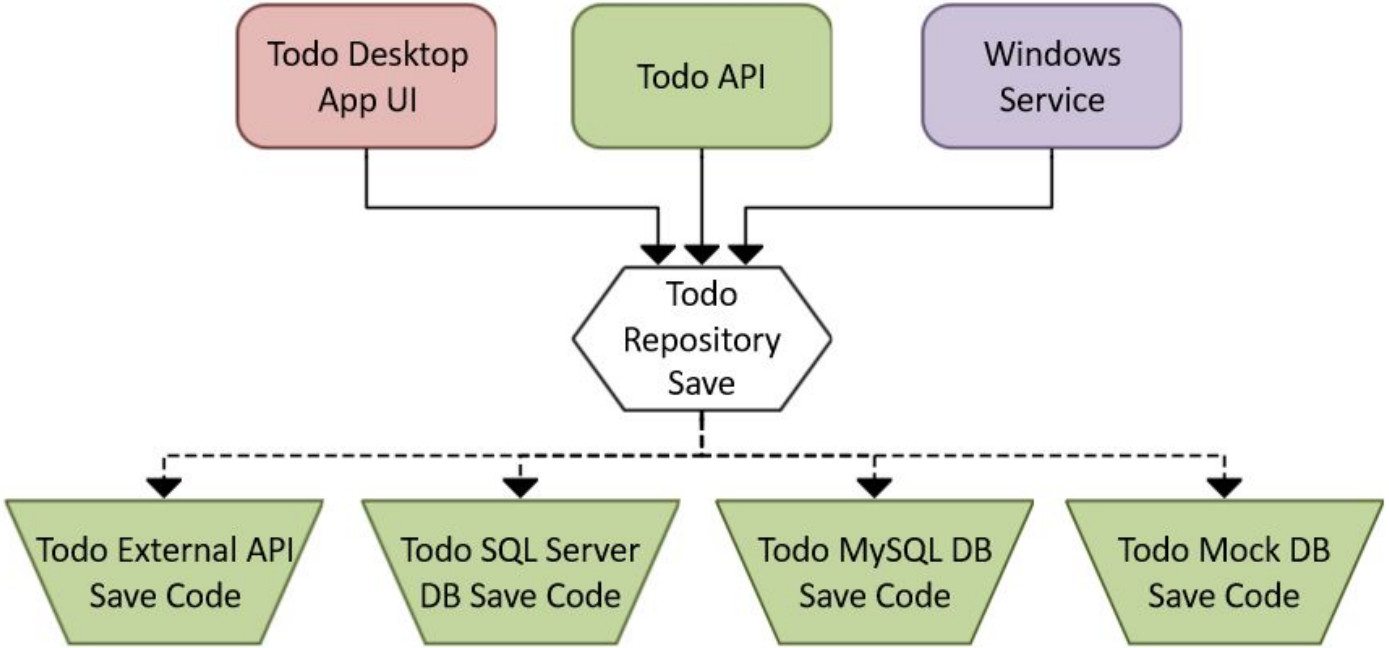
Now it can be used like any other service!

Replace your DbContext with the DbRepository in your controller

Without The Repository Pattern



The Repository Pattern



LINQ

Entity Framework Core uses Language Integrated Query (LINQ) to query data from the database. LINQ allows you to use C# to write strongly typed queries based on your derived context and entity classes.

```
// Specify the data source.
int[] scores = new int[] { 97, 92, 81, 60 };

// Define the query expression.
IEnumerable<int> scoreQuery =
    from score in scores
    where score > 80
    select score;

// Execute the query.
foreach (int i in scoreQuery)
{
    Console.Write(i + " ");
}
// Output: 97 92 81
```

LINQ Documentation

Working With LINQ

- Work with your persisted entities using LINQ as if they were simply in a collection.
- EF Core does the work of translating your LINQ expressions into SQL queries to store and retrieve your data.

Query syntax:

```
var linqExperts = from p in programmers
                  where p.IsNewToLINQ
                  select new LINQExpert(p);
```

Method syntax:

```
var linqExperts = programmers.Where(p => IsNewToLINQ)
                             .Select(p => new LINQExpert(p));
```

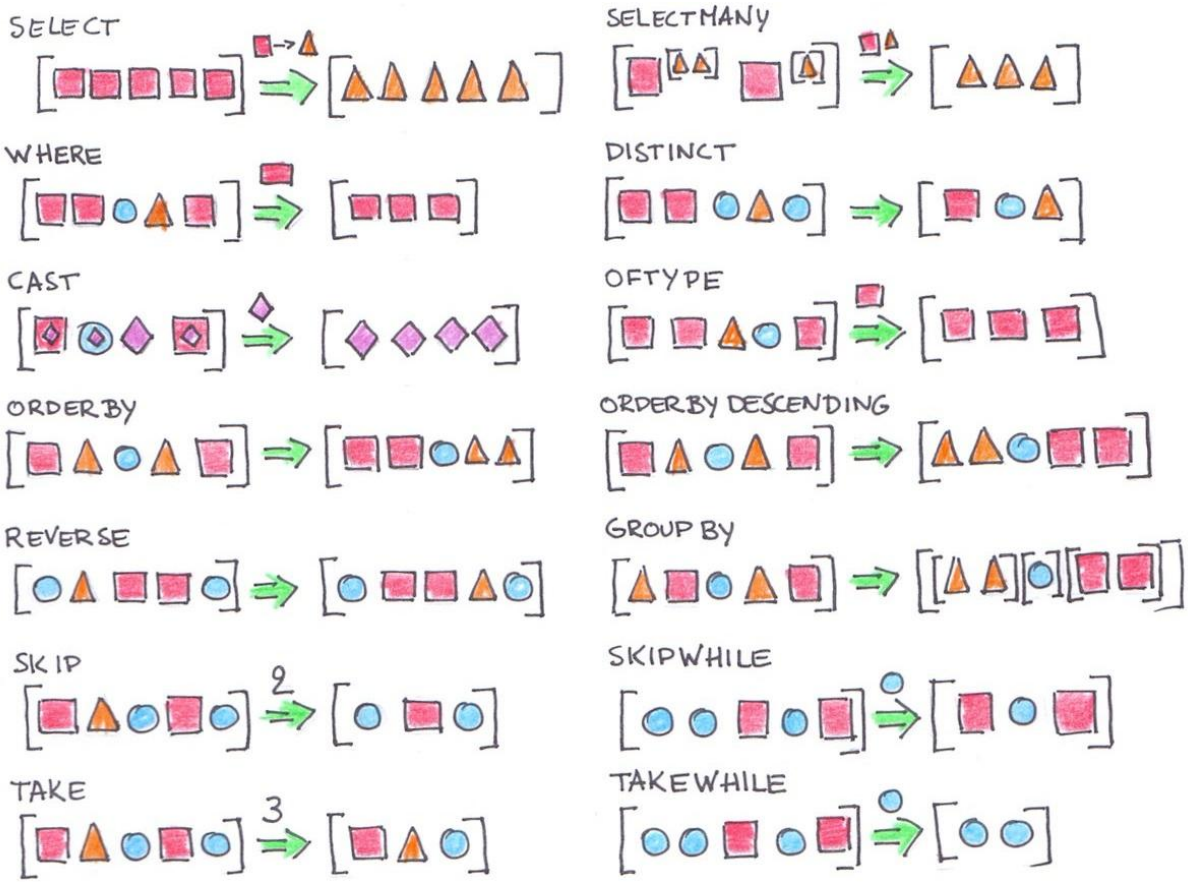
Using LINQ

LINQ Method Syntax


```
using (var context = new BloggingContext())
{
    // Loading all data
    var blogs = context.Blogs.ToList();

    // Loading a single entity
    var blog = context.Blogs
        .Single(b => b.BlogId == 1);

    // Filtering
    var blogs = context.Blogs
        .Where(b => b.Url.Contains("dotnet"))
        .ToList();
}
```

101 LINQ Examples

LINQ Explained To A Five Year Old



Exercises

Installations:

SQL Server Express

SQL Server Management Studio (SSMS)

SQLite

DB Browser for SQLite