ENTITY FRAMEWORK CORE & LINQ

ORM

- Programming technique for automatic mapping data and database schema
- Map relational DB tables to classes and objects
- ORM creates a "virtual object database"

OBJECT-RELATIONAL MAPPING

EF CORE

- **Entity Framework Core is the standard ORM framework for .NET Core**
  - Maps relational database to C# object model
  - Abstracts the underlying data provider
  - Powerful data manipulation API over the mapped schema
  - CRUD operations and complex querying with LINQ
  - Cross -platform

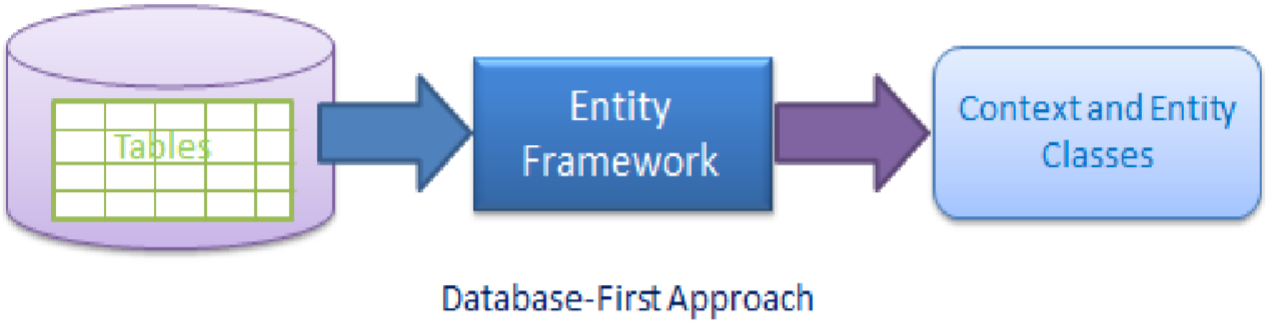
ENTITY FRAMEWORK CORE

- Works with a variety of database servers (Microsoft SQL Server, Oracle, SQLite, PostgreSQL, Cosmos DB)
- Rich mapping engine handle real-world database and work with stored procedure
- Generates strongly typed entity objects that can be customized beyond 1-1 mapping
- Translates LINQ queries to database queries
- Tracks changes, generating updates/inserts
- Materializes objects from data store call

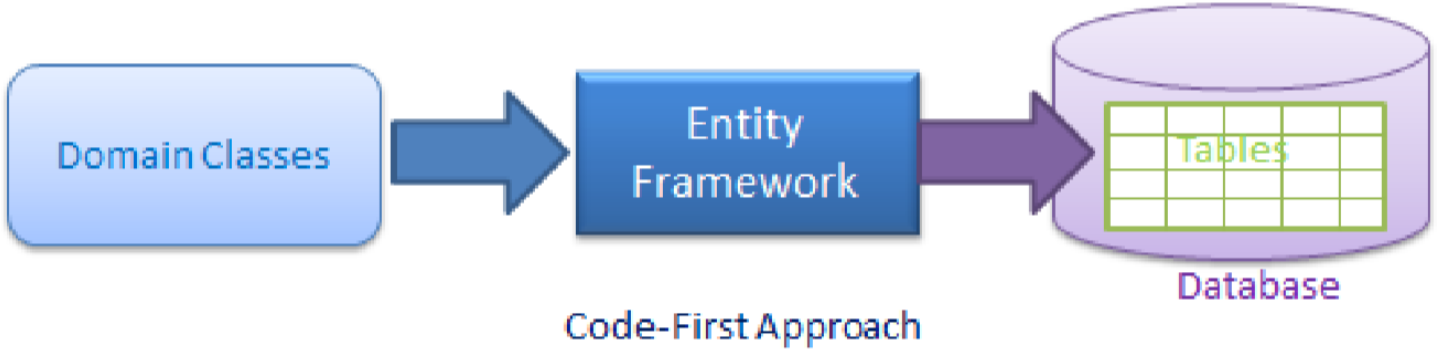
FEATURES

- Reduced development time
- Abstracts underlying database(it doesn’t matter what it is)
- Free from hard-coded dependencies on a particular data engine
- Mappings can be changed without changing the application code

BENEFITS



DATABASE FIRST


Domain Classes
Entity Framework
Tables
Database
Code-First Approach

CODE FIRST

EF COMPONENTS

- SqlServer
- SqlLite
- In-Memory
- 3rd party

DATABASE PROVIDERS

- Represents a session with the database
- Needs to be extended by your own context class
- It's like your own database in code holding a collection of DbSet<T>

DBCONTEXT


```
public class CarContext : DbContext
{
    public DbSet<Car> Cars{ get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlite("Data Source=cars.db");
    }
}
```

DBCONTEXT EXAMPLE

- Represents a collection of entities of a given type
- Used to query and save instances of entities
- **Linq** statements against a DbSet are translated into queries against the data store
- Note:
  - Items changed, removed or added are not persisted until **SaveChanges()** is called

DBSET<T>

ENTITIES

- POCO classes
- Mapped to relational data trough configuration
- Relate to other entities trough navigation properties

WHAT ARE THEY

- EF uses a set of conventions to build a model based on entity classes
- You can use Data annotation to override or hydrate entity classes or Fluent API
- Fluent API has highest precedence and will override conventions and annotations

HOW DOES ENTITY FRAMEWORK KNOW TO MAP ENTITIES?


```
class CarContext : DbContext
{
    public DbSet<Car> Cars{ get; set; }

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

MODEL WITH FLUENT API


```
public class Car
{
    public int CarId{ get; set; }
    [Required]
    public string LicensePlate{ get; set; }
}
```

MODEL WITH DATA ANNOTATIONS

- Primary key:
  - By convention, a property named **Id** or **<type name>Id** will be configured as the primary key of an entity
- Foreign key:
  - By convention if you have a navigational link between two entities (in Car you have a reference to User and vice versa) the dependent entity will have a FK if it contains a property named
  - **<primary key property name>**
  - **<navigation property name><primary key property name>**
  - **<principal entity name><primary key property name>**

CONVENTIONS

EF takes Id as PK by default

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

EF takes MyCustomKeyas PK

```
public class User
{
        [Key]
        public int MyCustomKey { get; set; }
        public string Name { get; set; }
}
```

CONVENTION PK OR DATA ANNOTATIONS PK

EF takes UserId as FK by default

```
public class Car
{
        public int UserId { get; set; }
        public User User { get; set; }
}
```

EF takes MyCustomFK as FK

```
public class Car
{
        public int MyCustomFK { get; set; }
        [ForeignKey("MyCustomFK")]
        public User User { get; set; }
}
```

CONVENTION FK OR DATA ANNOTATIONS FK

[Table]

[Column]

[Key]

[ForeignKey]

[NotMapped]

[DataType]

[Required]

[StringLength]

[DisplayFormat]

DATA ANNOTATIONS


```
var user = new User {Name = “Mercedesa"};
context.Users.Add(user);

context.SaveChanges();
```

CREATING NEW DATA


```
var user = context.Users.Find(1);
user.Name = “Jandarmeria“;

context.SaveChanges();
```

UPDATING DATA


```
var user = context.Users.Find(1);
context.Users.Remove(user);

context.SaveChanges();
```

DELETING DATA

```
var user1 = new User {Name = “Mercedesa"};
context.Users.Add(user1);
var user2 = new User {Name = "Jandarmeria"};
context.Users.Add(user2);
var user3 = new User {Name = “Seifa"};
context.Users.Add(user3);

context.SaveChanges();
```

IMPORTANCE OF SAVE CHANGES (THIS IS ALL ONE TRANSACTION)

- Used to create code from an existing database
- dotnet efdbcontextscaffold [datasource][dbprovider] -context<Name> --data-annotations --output-dir<Path>
- Example:
  - dotnet ef dbcontext scaffold“Server=(localdb)\mssqllocaldb;Database=Blogging;Trusted_Connection=True;”Microsoft.EntityFramework Core.SqlServer--contextCarContext--data-annotations --force -output-dir Data/Entities

SCAFFOLDING – FOR DATABASE FIRST

- Used to create/update a database from existing code
- In CLI in the folder of the project containing the DbContext:
  - dotnet ef migrations add InitialCreate
  - dotnet ef database update

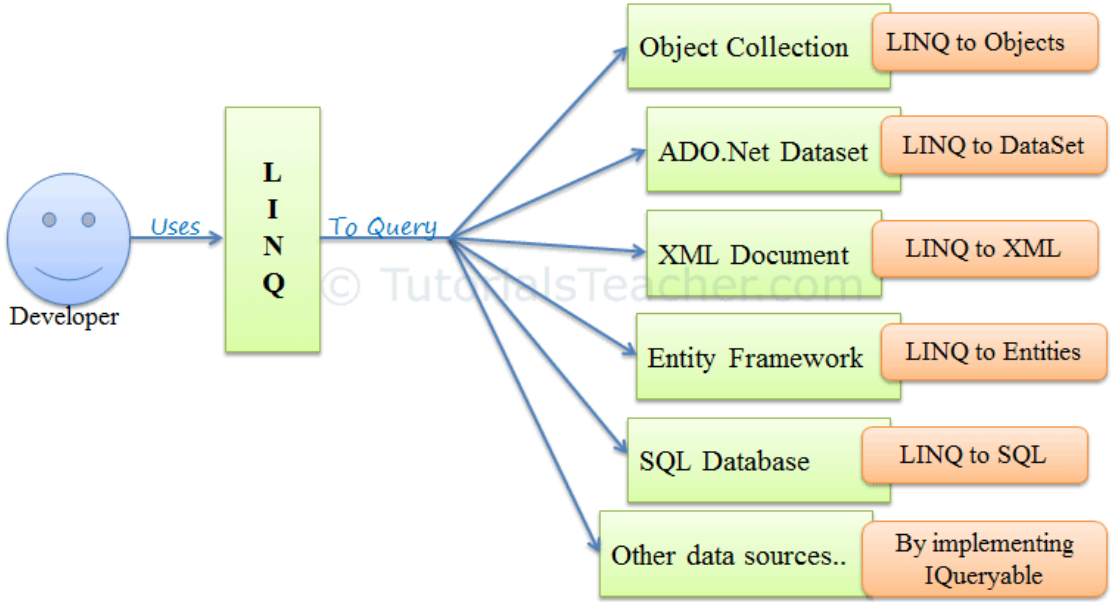
MIGRATIONS – FOR CODE FIRST

endava

LINQ

- Query capabilities directly into the C# language
- Provide a single querying interface for different types of data sources

LANGUAGE INTEGRATED QUERY

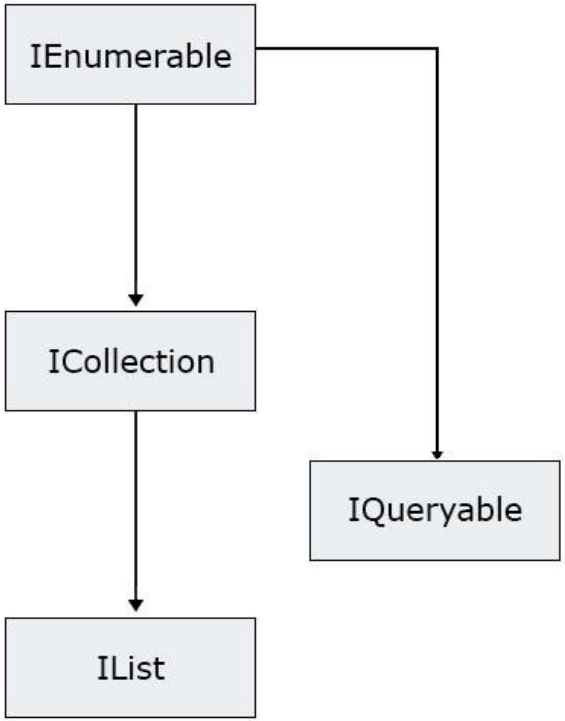


- Queries return results as objects
- Abstracts the transformation of different formats into objects

LANGUAGE INTEGRATED QUERY

- **IEnumerable** is just a container for elements
- **ICollection** supports operations like add, remove, update
- **IList** supports operations at different positions / foreach loop
- **IQueryable** executes LINQ directly over the database layer

COLLECTIONS



INHERITANCE

Thank You!

MATEI MADALIN