Step 5: Managing Changes with Migrations

- Now you have a DbContext, named AppDbContext, registered as a scoped service with the DI, and a data model corresponding to the database.
- Use migrations to generate SQL statements to keep your database's schema in sync with application's data model.
- Migrations provide a way to manage changes to a database schema when EF Core data model changes.

28

What is Migration?

- A migration is a C# code file in application that defines how the data model changed—which columns were added, new entities, and so on.
- Migrations provide a record over time of how database schema evolved as part of application, so the schema is always in sync with your app's data model.

29

How to Create Migration?

- Use command-line tools to create a new database from the migrations or to update an existing database by applying new migrations to it.
- The EF Tool package is used to manage databases from the command line and to manage packages for the project that provide data access.

30

EF Tool (一)

- Command to install and uninstall the tool package.

```
> dotnet tool uninstall --global dotnet-ef
> dotnet tool install --global dotnet-ef --version x.x.x
```

31

EF Tool (二)

```
> dotnet ef --help
```

```
Entity Framework Core .NET Command-line Tools 7.0.4

Usage: dotnet ef [options] [command]

Options:
  --version        Show version information
  -h|--help        Show help information
  -v|--verbose     Show verbose output.
  --no-color       Don't colorize output.
  --prefix-output  Prefix output with level.

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

Use "dotnet ef [command] --help" for more information about a command.
```

32

EF Tool (三)

- Create a migration.

```
>dotnet ef migrations add InitialSchema
```

- This command creates three files in the Migrations folder in project:
  - Migration file
  - Migration designer.cs file
  - AppDbContextModelSnapshot.cs

33

Migrations Folder（一）

- Migration file with the Timestamp_MigrationName.cs format, describes the actions to take on the database, such as creating a table or adding a column.
- Note that the commands generated here are database-provider-specific, based on the database provider configured in project.

34

Migrations Folder（二）

- Migration designer.cs file describes EF Core's internal model of data model at the point in time when the migration was generated.

35

Migrations Folder（三）

- AppDbContextModelSnapshot.cs file describes EF Core's current internal model.
- This file is updated when adding another migration, so it should always be the same as the current (latest) migration.
- EF Core can use AppDbContext-ModelSnapshot.cs to determine a database's previous state when creating a new migration without interacting with the database directly.

36

EF Tool（四）

- Adding a migration doesn't update anything in the database itself.
- You must run a different command to apply the migration to the database.

```
>dotnet ef database update
```

37

EF Tool（五）

- The "database update" command performs four steps:
  1. Builds the application;
  2. Loads the services configured in app's Program.cs, including AppDbContext;
  3. Checks whether the database in the AppDbContext; connection string exists and if not, creates it;
  4. Updates the database by applying any unapplied migrations.

38

EF Tool（六）

- Display the SQL commands that the migration will execute in the database.

```
>dotnet ef migrations script
```

- Remove the most recent migration added to the project

```
>dotnet ef migrations remove
```

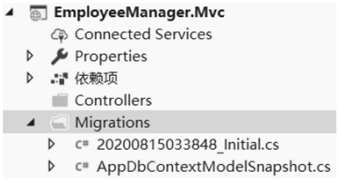
- Drops the database completely

```
> dotnet ef database drop --force
```

39

Project: Creating the first migration

```
> dotnet ef migrations add Initial
```

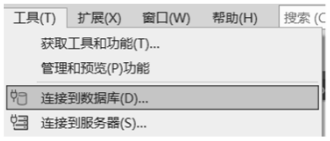



```
> dotnet ef database update
```

40

Project: Check the database created by EF（一）

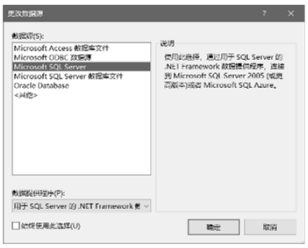
■Select the menu in Visual Studio.



41

Project: Check the database created by EF（二）

■Select Microsoft SQL Server as the data source.



42

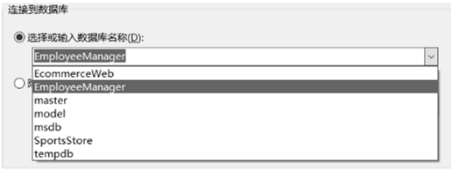
Project: Check the database created by EF（三）

■Input the database server name.

43

Project: Check the database created by EF（四）

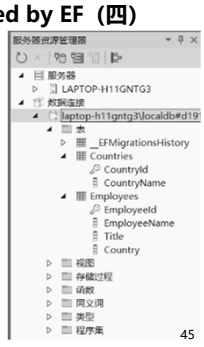
■Select the database you'll check on the server.



44

Project: Check the database created by EF（四）

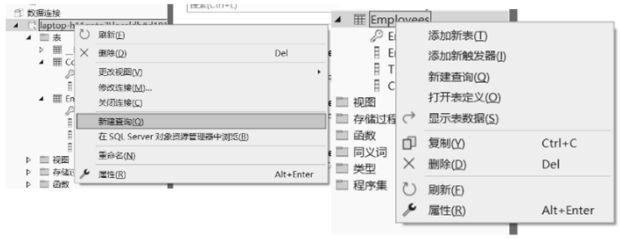
■It shows the database schema.



45

Project: Check the database created by EF（五）

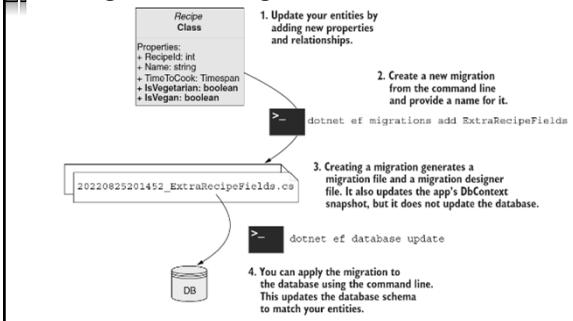
- You can make a query or other operations on the database.



Adding a Second Migration（一）

- Most applications inevitably evolve due to increased scope or simple maintenance.
  - Add properties to entities;
  - Add new entities;
  - Remove obsolete classes;
  - ……

Adding a Second Migration（二）



3. Data Operations

Querying Data from the Database

- The key to the data operations is the DbSet<T> class.
- It is used as the result for the properties defined by the database context class.

```
public class AppDbContext:DbContext
{
    ……
    public DbSet<Employee> Employees { get; set; }
    public DbSet<Country> Countries { get; set; }

}
```


Reading an Entity by Key

- Find(key) method reads the row in the table that has the specified key and returns an object that represents it.

```
context.Employees.Find(id);
```

Querying All Entities

- To retrieve all the data objects stored in the database, read the value of the context class's Employees property, which returns a DbSet<Employee> object.
- Entity Framework Core doesn't read the data from the database until you enumerate the DbSet<T> property.

```
return context.Employees;
```

52

Project: Create the repository interface

- Create two repository.

```
public interface IEmployeeRepository
{
    List<Employee> SelectAll();
}
```

```
public interface ICountryRepository
{
    List<Country> SelectAll();
}
```

53

Project: Implement the repository interface

```
public class EmployeeRepository:IEmployeeRepository
{
    private AppDbContext context;
    public EmployeeRepository(AppDbContext ctx)
    {        context = ctx;    }
    public List<Employee> SelectAll()
    {        return context.Employees.ToList();    }
}
```

```
public class CountryRepository:ICountryRepository
{
    private AppDbContext context;
    public CountryRepository(AppDbContext ctx)
    {        context = ctx;    }
    public List<Country> SelectAll()
    {        return context.Countries.ToList();    }
}
```

54

Project: Configure the application

- AddTransient method ensures that a new Repository object is created each time a dependency on the IRepository is resolved.

```
services.AddTransient<IEmployeeRepository,EmployeeRepository>();
services.AddTransient<ICountryRepository, CountryRepository>();
```

55

Project: Create a controller

```
public class EmployeeManagerController : Controller
{
    private IEmployeeRepository repo;
    public EmployeeManagerController(IEmployeeRepository r)
    {
        repo = r;
    }
    public IActionResult List()
    {
        return View(repo.SelectAll());
    }
}
```

56

Project: Add a List View

```
@model IEnumerable<Employee>

<h2>Employee Info List</h2>
<table>
    <tr>
        <td>Employee ID</td>
        <td>Employee Name</td>
        <td>Employee Title</td>
        <td>Employee Country</td>
    </tr>
    @foreach (Employee e in Model)
    {
        <tr>
            <td>@e.EmployeeId</td>
            <td>@e.EmployeeName</td>
            <td>@e.Title</td>
            <td>@e.Country</td>
        </tr>
    }
</table>
```

57