

.NET Core 6 Entity Framework: How-To Guide for Professionals - Summary



Course Description

Welcome to .NET Core 6 Entity Framework: How-To Guide for Professions. This course is designed to incrementally walk you through the best practices and techniques used to build robust and extensible software application based on the latest Microsoft data access technology, .net Core 6 Entity Framework. We will demonstrate the latest capabilities and features, as well as review and present common design patterns and their implementations. This guide is designed to helps experienced and novice software developers navigate the new and sometime subtle feature in .net Core 6, as well as lay the foundation understanding for engineers learning this paradigm.

We will focus on the inner workings of the Data Service layer of a sophisticated (imaginary) online book store start-up, looking at an aspect of managing their libraries of books (outlets) and their customers. Throughout this course, we will construct C# classes that support and demonstrate each lesson as well as test the code through design-time invocations and build test cases to verify the run-time environments. Not to worry – these concepts will also be explained.

We will start from scratch, learning the types of project and components needed to create classes that represent persisted object stored in a database. You will learn to describe those classes, and understand the concept of models. You will learn to create and enhance features that automatically create, manage changes, and map classes to those backend database entities; known as ***Migrations***. Throughout this course, we will use Migrations to effect changes to the back-end repository and verify our code through our unit tests.

We will discuss the role of the ***DbContext*** within the Entity Framework, and demonstrate how to use it with ***Data Annotations*** and the ***Fluent API*** to better describe classes and enhance their behaviour, interaction and relationships with other entities. You will learn what each are best used for, how they effect migrations and the unlaying database tables and columns, and deploy those change via migrations to the backend repository. You will learn how to define indexes on database table, understand the different types of relationships between classes and create navigation routes through them.

Once we’ve gotten that under out belt, we’ll dive into database operations to support all aspects of data management – create, read, update and delete (***CRUD*** ***Operations***). We will review best practices and performance concern, look at patterns and techniques that reduce technical debt. You will learn the difference and when you use / support ***IEnumerable*** and ***IQuerable*** interfaces, as well as how to read related data. We review software layering concepts, and how to avoid “***Boundary Bleed***” and the ***Sink-hole anti-pattern***.

We will also explore and demonstrate reverse engineering a database (schema) into generated classes that map to those back-end database entities (***Scaffolding***). You will learn and use a sophisticated real-work implementation of the ***Repository*** and ***Unit Of Work*** patterns, as well as learn advanced concepts such as ***Paged Result*** Sets that provide performant result to end-users when there is vast amounts of potential data. We will wrap-up with showing how to use Custom SQL (***Raw***) to perform sophisticated query and other operations.

Each section is packed with incrementally advancing lessons, supported by code starting point and snapshots along the way. If you get stuck, not to worry – we have code that gives you a fresh start at the beginning of each section.

What Will You Learn?

* **Migrations – Code Generation, Class Mapping, Schema Generation,**
* **Migrations - Deployment Scenarios, Design-Time vs Run-Time, and Project Organization**
* **Data Annotations – Class Mapping, Schema Definition, Reduce Technical Debt**
* **Fluent API – Class Mapping, Performance, Schema Definition (Db Generation)**
* **CRUD Operations – Data Management, Patterns, Code Quality**
* **Repository Pattern – Design Patterns & Techniques, Performance, Reduce Technical Dept**
* **Unit Of Work Pattern –Design Pattern & Techniques, Transactions, Reduce Technical Dept**
* **Reverse Engineering (Scaffolding) – Code Generation, Automation**
* **Paging Data – Performance, Scalability**
* **Asynchronous Operations – Performance, Scalability**

Course Roadmap

The course begins with a thorough examination and demonstration of Migrations, from why use them to sophisticated scenarios where a custom project and multiple database providers / domain contexts are considered. We will then look at Data Annotations, and how they are used to describe classes and the impact on Migrations. Then we’ll learn how the Fluent API can be used to achieve even more sophisticated mappings and schema alterations. From there, we’ll take a look at software design patterns related to data operations, and create formal implementations of those patterns and interfaces, examine design principles and best practices. We will then look at advanced concepts such as mapping and making use of SQL Views, Scaffolding (reverse engineering), and performance enhancements for searches using paged data and asynchronous operations.

Chapter-wise Description

# Chapter One - Migrations

## Why – Purpose and Usage

In Chapter One we begin our exploration of .net 6 Core by starting with a description and overview of the code-first metaphor, and how Migrations within .net 6 Core framework are design to support it. We explain what migrations are and how to use them to create database entity that support our data models (C# classes). We will explore what migrations are, why they are useful and how to best use of the concept and mechanics.

## Classes and Relationships

We will then discuss and define various types of relationships between classes and the database entities they represent. You will learn how to use migrations to establish database entities such as tables and their primary key and foreign keys and how those can be used to navigation the object model. We will walk you through an evolution of the object model from a Code-First perspective, demonstrating the application of Migrations to synchronize changes to the back-end repository (database schema).

## Design-Time DbContext

Through our exploration of Migrations, we learn about the DbContext and how it is used at Design-Time to push changes to our class definitions (object models) into the definition of their back-end database entities (tables, views, indexes, keys, and stored procedures).

## Manage Migrations

We will walk through several iterations of code changes to our model and how we use Migrations to keep our repository in synch. You will learn what to watch out for when applying changes to model and pushing them out to the schema, and how to revise the migration code to effect specific changes and avoid side-effects.

## Synchronization of model and database schema

We will explore the process of Local Db Synchronization vs the use of Production application scripts, bundling Database change deployments through generated exe, and applying migrations selectively.

Extend Migrations

You will become familiar with how to supply custom parameters, new operations, and invoke extension methods during migration process.

Custom projects

You will get hands-on practical example of moving Migrations into a separate project to house migrations exclusively. We will show you how to manage multiple providers and ***DbContext*** derived classes, create multiple design-time ***DbContextFactory*** instances, and perform migrations on each.

# Chapter 2 – Data Annotations

## Entity / Db Design

In Chapter Two, we turn our focus to Data Annotations – with an explanation of how to use them and the various ways they are used to better describe our classes and the resultant effect on the generate database entities.

## Code First

You will lean how to make fine-tune adjustments to our class models through the use of Data Annotations to target specific implementations of data type and size for table columns mapping to class members. We will demonstrate how to change the names of tables and column create in the back-end repository so that they differ from their class name and property counterparts. You will learn how to use conventions (patterns in class / property naming) to automatically establish database constructs and behaviour.

## Subset of Configuration Options

We will explore and make use of most of the available configuration options available through Data Annotations. Complex relationships will be demonstrated, such as when two entities have more than a single relationship. You will learn about the class identifiers and how they can be defined as primary keys and generated as GUID’s, sequential integers (Identity), or computed (custom calculation). We will be generating and applying migrations at each step of our lesson journey.

## Class and Property Annotations

We will make use of most of the available Annotations applicable to Classes and their member Properties. Using Migrations, those annotations will be used to further enhance the storage structures, indexing, and relationship between database entities, and you will learn how to best use these to make sure your application is performant.

# Chapter 3 – Fluent Interface

## What Is it? Overview

Chapter Three begins with an introduction to the Fluent design Pattern (also known as the Fluent API). We will explain what it is and how it is used in our class models to affect a mechanism of method chaining, that can make our code more easily understood and readable (like a natural language). We will demonstrate how to use the ***ModelBuilder*** class, which provides a Fluent API implementation in .net entity framework, to do everything and more than what is available using Data Annotations.

## Entity Mapping

Using the Fluent API, we will demonstrate how to override the default behaviour and those prescribed by Data Annotations to effect changes to the underlaying database table, its structure and schema membership. We will show how the Fluent API takes precedence over Data Annotations, and how it is much more powerful and robust. You will learn how to control the type of primary key and its initialization, how to define alternate keys and Indexes for improved query performance.

## Property Mapping

Next you will learn how to use the ***ModelBuilders’*** Fluent API to control the database column mappings to class Properties and override the column name, data type, and sequence withing the table structure. We will show you how to set the nullability of a column as well as its default value, and size. We will discuss concurrency models and show you how you can use it to control Update and Delete operations in a multiuser transactional environment.

## Relationship Mapping

We will then review the different types of relationships between entities and how they are supported by the Fluent API as compared to Data Annotations. We will take a close look one-to-one, one-to-many, and many-to-many relationships and how they are implemented and supported by the .net core 6 entity framework. You will learn to review generated migration code and revise it for special relationship-based behaviours, such as cascade delete or set references to null.

## Stored Procedure Mapping

What if you need to create stored procedures to perform special processing within the database or complex queries? You will learn just how easy it is to create stored procedures through migrations and map the results of that SQL code to a Class and its member Properties.

## Raw SQL Mapping

To further the customizations possible, you will learn how to craft custom query and create a specialized class structure to contain and map to its results, without the use of stored procedures. You will also be shown that you can use this mechanism to run a query that perform a data operation by does not return any data results – just the relative success or failure of that operation.

# Chapter 4 – CRUD Operations

## Understanding DbContext

Now that we’ve become well versed with the *Design-Time* usage of the ***DbContext*** through ***migrations*** to change the structure and mappings of classes to database tables, we shift our focus to the *Run-Time* aspect of the ***DbContext*** and how we use it to access and manage persisted data (rows of data in database tables). You will learn the optimal use of the ***DbContext*** and its designed lifetime. We will demonstrate best practices and implementation details that provide optimal performance and scalability. You will learn how to use asynchronous operations to improve overall application performance by decoupling the connection with long running database queries. You will see how the .net core 6 entity framework performs change tracking withing the context of its lifetime, and how we can supply options on the ***DbContext*** derived classes’ constructor for Inversion-Of-Control (IoC) supported run-times (such as asp.net core 6).

## Creating & Persist Data

We introduce the concept of a Repository for each database entity (table) that is based on the DbContext. You will learn how these objects are constructed and how to implement methods that create and store new instances within the backend database. You will see the default mechanism for unique ID creation for new entities, as well as creating related entities within the same process. You will be shown how to create Unit Tests to verify our code and ensure this code is fully operational.

## Retrieving Data

Once we’ve demonstrated and tested how to insert data, we need to be able to retrieve it using its unique identifier as well as other criteria. We will look at the various means to find and return a specific row of data or a collection of data that map to instances of our model classes. You will be introduced to the synchronous and asynchronous design patterns and how to implement and test them.

## Updating Data

The next thing in our itinerary is learning how to update instances of data stored in our repository. We will further extend our implementation to show how we use our repository can be used to update data and verify that the update was successful. We will discuss how the entity framework keeps track internally on changes to entities, and how we can selectively apply changes to specific object properties.

## Removing Data

There are several mechanisms to remove data instances from the back-end repository, and we will demonstrate and test each one. We will walk through how delete functions within the entity framework and how that effect change-tracking. You will lean how to create validation tests to confirm the correct operation of our removal functions.

## Accessing Related Data

Often when we are looking for data from a well designed (normalized) database, we need to pull in related data to the primary entity we are searching for. We will demonstrate how to achieve this in both synchronous and asynchronous forms. You will lean this can be done for retrieval of a single primary instance and all its related data, as well as return a set or search results and its child instances.

## Searching for Data

In the final lesson of this section, we will review how to search for data from our repository and return fully populated lists of mapped class instances. Building on what we’ve learned so far, we will also demonstrate synchronous and asynchronous retrieval mechanisms, as well as returning data that is related to the primary entity being searched.

# Chapter 5 – IQueryable vs. IEnumerable

## IQueryable Explained and Usage

When we search for data using the Entity Framework, there are a number of interfaces that we can use to formulate the resultant query and data. The distinction between the two is important, as it defines the boundary between the search conditions, and the results of that query. We will explore how we can use and benefit from the IQueryable Interface to build flexible and complex search conditions.

## IEnumberable Explained and Usage

As a standard interface for lists, this interface is a common return for collections of Entities. We will compare and contrast IEnumerable with IQueryable and learn when to use each, as a best practice. We will explore the performance benefits of each, and how to make the best performant queries that return only the data to your client application needs. We will explore and demonstrate pagination, showing the performance improvements when only gathering the data that is going to presented within the end-user interface. We will use our test applications to validate and demonstrate, and compare the interfaces.

## Reading Related Data

In this video, we will demonstrate and show how to use keywords that enable us to search for entities related to our primary entity of interest; applicable to both IEnumerable and IQueryable. We will show you how to create methods to locate information from the primary table (entity) of interest, and those that hare a referential relationship with it (as established through previous exercises). You will see how to return data in a child-parent relationships, as well as foreign key information, and validate that through unit tests.

## Software Design Patterns, Layering, and Boundary Bleed

In this video, we dive into design concepts and why and how we can build applications that are layered and independent (loosely coupled), and how we use IEnumerable and IQueryable interfaces to implement those. The various layers of an application will be explained and why we have them, demonstrate how to implement and message flow, as well as the patterns (and Anti-Patterns) to be aware of. We will explore and conceptualize the layering of an application and rational to how information and requests flow through it.

# Chapter 6 – Advanced Concepts, Patterns, & Implementations

In this final section, section 6, we weave together all the lessons learned thus far and complement them with the next layer of advanced concepts, design patterns and possible implementations with full featured examples.

## Accessing Sql Views

You will be shown and learn how to design and define Sql Views, from a code-first perspective; define them in code, test drive the sql script to create them and how to integrate their definitions within the Migrations mechanism. You will learn how to map their resultant columns to class properties, create repository for managing access and test harness to prove and validate your Sql View from test project code.

## Reverse Engineering

So far, we have been building an application from ground up. What if you are starting with an existing database schema and a database fully (or not) populated with data? This leads to the other side of the coin, and an additional development scenario; an existing database design or a separate development group that focus entirely on the data structure and software development teams worked dependent on their output. In this case, we need to revere-engineer their database schema design into an appropriate class structure for use by our applications. The .net entity framework supports this scenario, called “Scaffolding”, and we will explore all the capabilities within this feature and how to make use of it and all of its options.

## Repository Pattern

So far we have touched on the idea of a Repository Pattern and how to use it to encapsulate database entity operations. We will take this concept further and explore and demonstrate and fully fleshed out implementation of both a standard repository interface for all database entity repositories as well as base class implementation. You will see how this pattern and the implementation we create can help build robust business applications faster and more reliably. You will see how to rapidly create tests to validate all aspects of our model, to simplify automated testing, improve code quality while reducing technical dept.

## Unit-Of-Work Pattern

Continuing on with the use of patterns to improve our applications, we then move onto a discussion of the Unit-Of-Work pattern and how it works in conjunction with the Repository Pattern. We will explore and come to a better understanding of transaction scope and how that relates to operations on our entities and making those changes persistent (at the database) and consistent (within each open transaction). You will learn about database scope of processing (the “transaction”) and how that relates to isolation of operations.

## Search & Paged Results

To help make our applications more performant, we must provide intelligent interfaces that provide just the right amount of data that is readily consumable. This means providing the ability to search for specific data items using a number of potential matching criteria, as well as limiting the set of data return while still being able to navigate through the entire result set if desired. You will see how we can extend the base class and interfaces for the repository pattern to enable pagination and search across all entities in our model.

## Raw Sql

To wrap up our tour of the .net 6 Core Entity Framework, we will go full custom with Raw SQL queries and show how we can build them safely with user supplied criteria to avoid Sql Injection Attacks. You will learn how to build your custom Sql into Migrations and thereby establish a relationship with database entities, and output based on the mappings to them.

Conclusion

This course covers the .net 6 core entity framework from the basic understanding and concepts, and builds upon that adding layers of knowledge and capability with each section. These are hand-on lessons that walk you through building and testing a live example with actual unit tests and deployments to our target database. We will go from design-time to run-time, and demonstrate how to test and validate each.

Once you have concluded this course, you should be confident and well versed in all aspects of the .net core entity framework and ready to build your next business application back-end data access layer.

Next Steps

Once completing this course, you will have a concreate footing to stand on when discussing and developing data service layers and prepare you for discussions on the middle tier (business logic layer) and the front-end (presentation layer). Your knowledge gained through this course will help you build better applications and possibly explore development of those tiers and the technologies best suited. Watch for other course coming soon – End-to-End Security (for web, mobile, desktop, and azure), Front-End Development with Blazor (Client & Server), and Business Tier Methods, Practices & Patterns.