Code-First Development with Entity Framework

Entity Framework Code-First enables developers to read and write data in a relational database system using C# or VB.NET. It is Microsoft’s answer to demand for an ORM from .NET developers.

This book will help you acquire the necessary skills to program your applications using Entity Framework. You will start with database configuration and learn how to write classes that define the database structure. You will see how LINQ can be used with Entity Framework to give you access to stored data. You will then learn how to use Entity Framework to persist information in a Relational Database Management System. You will also see how you can benefit from writing ORM-based .NET code. Finally, you will learn how Entity Framework can help you to solve database deployment problems using migrations.

Who this book is written for

This book is intended for software developers with some prior experience with the Microsoft .NET framework who want to learn how to use Entity Framework. This book will get you up and running quickly, providing many examples that illustrate all the key concepts of Entity Framework.

What you will learn from this book

- Understand what ORM tools are and what problems they solve
- Set up a .NET application to use Entity Framework
- Write Entity Framework code that creates updates, and deletes application data in a database
- Learn how to query the data using LINQ and gain insight into how LINQ fits into Entity Framework
- Create a database structure using the Entity Framework configuration API
- Evolve the database structure using migrations
- Write asynchronous code and handle concurrency
- Work with stored procedures and functions using Entity Framework

In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 01 'Introducing Entity Framework'
- A synopsis of the book’s content
- More information on Code-First Development with Entity Framework

About the Author

Sergey Barskiy is an architect with Tyler Technologies. He lives in Atlanta, GA. He has been developing software for almost 20 years. Sergey is a Microsoft MVP. He holds these Microsoft certifications: MCPD, MCTS, MCSD for .NET, MCAD for .NET, MCDBA, and MCP. He has been working with Microsoft Technologies for over 15 years. He is a frequent speaker at various regional and national conferences, such as VS Live, DevLink, CodeStock, and Atlanta Code Camp, as well as local user groups. He is one of the organizers of Atlanta Code Camp. He authored articles for Code Magazine.

Sergey Barskiy has been using Entity Framework since it was first released to the public. He has deployed a number of projects to production that used Entity Framework over the years. He has used the Code-First approach on a few different projects as well. Sergey has produced an online video training course for this technology. He has spoken on Entity Framework Code-First at a number of national and regional conferences and events.

You can tweet to him at @SergeyBarskiy or e-mail him at sergey@barskiy.com.

I would like to thank my family for putting up with my busy schedule during the time I was working on this book. I want to also thank Packt Publishing for giving me the courage and opportunity to work on this project.
Code-First Development
with Entity Framework

I have been writing applications on the Microsoft platform for almost 2 decades. Many, if not all of them, use databases to persist user data. I have used many technologies to access data, starting with ADO.NET. Object Relational Mapping (ORM) tools, have many advantages over ADO.NET. They allow developers to write data access code faster and safer. ORM tools have been designed to solve impedance mismatch problems between object-oriented programming and relational databases. Microsoft's Entity Framework is the company's answer to the demand for an ORM from .NET developers. This book is the guide that will help you acquire the necessary skills to program your applications using Entity Framework.

This book centers on the Code-First approach with Entity Framework, which has become the most common way of using the technology. Code-First allows developers to control the entire data access layer of their applications from the .NET code. This approach simplifies and streamlines the entire application development life cycle, keeping developers coding inside Visual Studio, the only tool they need to use Entity Framework.

The book starts with the basic concepts of defining the database structure via C# and VB.NET code, then progresses to full data access. Chapters cover create, read, update, and delete operations (CRUD) with Entity Framework. It also shows how to update the Relational Database Management System, (RDBMS) structure, via the migrations API. It explores aspects of data access in both .NET languages using the Languages INtegration Query (LINQ), API. Because of Microsoft's continuous commitment to both C# and VB.NET, the book contains examples in both languages in every chapter.

I have been using Entity Framework since 2008, and I felt that I had the necessary experience to write a book on the subject. I spoke on the topic on many conferences and events and saw tremendous interest in creating a concise guide to Entity Framework. This was one of my primary motivations in creating a shorter textbook. I read many technical books while working in the industry, and I myself, at times, had trouble maintaining the focus while reading 800-page technical books. They definitely have a place in the industry and are very useful. However, I feel they are intimidating for the developers who are just getting started with a particular technology. My hope is that this book will get you going quickly on the new topic and have you writing data access code in a few hours. You should be able to master the foundation behind Entity Framework with this book quickly and easily.
What This Book Covers

Chapter 1, *Introducing Entity Framework*, gives us an understanding of what the, Object Relational Mapping (ORM) technology brings to developers. You learn the history of Entity Framework as an example of an ORM. We study the architecture behind the Entity Framework technology.

Chapter 2, *Your First Entity Framework Application*, teaches us how to create our first project that uses Entity Framework. We create classes that map to database tables. We observe how our target database is created when the project is run. Finally, we save and retrieve our first data from the created database.

Chapter 3, *Defining the Database Structure*, dives deep into details of mappings between classes and tables. We create maps between properties to columns as well as rules that govern such mappings. We define relationships between classes that translate into relationships between tables. We exercise multiple approaches that can be used to define the mappings.

Chapter 4, *Querying, Inserting, Updating, and Deleting Data*, discusses how to use the LINQ API, that allows developers to retrieve the data from the database. We sort, filter, and perform element operations and use quantifiers. We query related entities. You learn the advantages and pitfalls of eager and lazy loading. We insert, delete, and update the data.

Chapter 5, *Advanced Modeling and Querying Techniques*, dives deeper into modeling and querying techniques. We use complex types to have more consistency in the database structures. We create an explicit table and column names. We define structures that use table and entity splitting. We use projections in queries to make them more efficient and summarize our data. We page the data for retrieval, breaking it up for presentation to the users. We use joins to create queries that use related entities.

Chapter 6, *Working with Views, Stored Procedures, the Asynchronous API, and Concurrency*, shows how to access with database views from Entity Framework. We query data via stored procedures using the Entity Framework API. We perform create, update, and delete operations with stored procedures. We exercise Entity Frameworks and the asynchronous API and learn the advantages and pitfalls of asynchronicity. We implement concurrency handling, learning to handle the situation when multiple users attempt to update the same data.
Chapter 7, *Database Migrations and Additional Features*, shows how to enable migrations on our Entity Framework project, creating and updating the database schema without data loss. We use implicit migrations first, then create explicit migrations, customizing our migration code. We use common aspects of the migrations API, adding columns and specifying default values. We apply migrations using multiple approaches. We create migrations from an existing database. We dive briefly into useful Entity Framework features, not covered previously.

*Appendix, Answers to Self-test Questions*, contains answers to questions you will find throughout the book.
Chapter 1

Introducing Entity Framework

In this chapter, you will be introduced to Entity Framework. You will gain an understanding of Object-Relational Mapping (ORM) tools and the problems they solve. A brief history of Entity Framework will also be covered in this chapter. We will examine the capabilities of Entity Framework and its architecture.

In this chapter, we will cover the following topics:

- ORM tools and the problems they solve
- A brief history of Entity Framework
- The capabilities of Entity Framework
- The overall architecture of Entity Framework

What is ORM?

When it comes to business software, almost all of it needs to store data that pertains to its functions. For many decades, Relational Database Management System (RDBMS) has been a go-to data storage for developers. ORM is a set of technologies that allows developers to access RDBMS data from an object-oriented programming language.

There are other RDBMSes available, such as SQL Server, Oracle, DB2, MySQL, and many more. These database systems share some common characteristics. Each system supports one or more databases. Databases consist of many tables. Each table stores data in a tabular format, divided into columns and rows. Data rows in multiple tables may relate to each other. For example, a person's details stored in the Person table can have phone numbers stored in a separate Phones table.
Introducing Entity Framework

In the following screenshot, you can see a table that allows you to store a person's information, specifically their first and last names, along with a unique identifier for each person. This type of storage, where similar data items are grouped together into tabular structures is typical:

<table>
<thead>
<tr>
<th>PersonId</th>
<th>FirstName</th>
<th>LastName</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>John</td>
<td>Doe</td>
</tr>
<tr>
<td>2</td>
<td>Jane</td>
<td>Williams</td>
</tr>
<tr>
<td>3</td>
<td>John</td>
<td>Smith</td>
</tr>
<tr>
<td>4</td>
<td>James</td>
<td>Johnson</td>
</tr>
</tbody>
</table>

Each column can also be constrained in some ways. For example, **PersonId** is an integer column. **LastName** is `nvarchar(50)` column, which means you can store Unicode data of variable size in it, up to 50 characters. You will see in subsequent chapters how we describe this information using Entity Framework.

The data stored in each column and row combination is scalar data, such as number or string. When software needs to persist or retrieve data, it must describe its intent, such as insert or select query, using the database-specific language called **Structured Query Language (SQL)**. SQL is a common standard for all relational database systems, as issued by the **American National Standards Institute (ANSI)**. However, some database systems have their own dialect on top of the common standard. In this book, we are not going to dive into the depths of SQL, but some concepts are important to understand. There are some basic commands that we need to look at. These are typically described as CRUD. **CRUD** stands for Create, Retrieve, Update, and Delete. For example, if you want to retrieve or query the data from the preceding example, you would type the following:

```
SELECT PersonId, FirstName, LastName
FROM Person
```

Historically, before tools such as Entity Framework, developers embedded SQL language statements inside the software code using .NET languages, such as C# or VB.NET or other programming languages, such as C++ or Java. The reason for this is that these languages do not natively speak or understand SQL. For example, to retrieve the data from the database and manipulate it as objects, you would write a fair amount of code using **ADO.NET**, .NET Framework's data access built-in framework. You would need to define a class to hold a person's data. Then, you would need to open a connection to the database, create a command that uses the preceding query as its text, execute the command's reader, and iterate through the reader results, populating an instance of our **Person** class with the data from the reader. As you can see, there would be a lot of steps involved. More importantly, the code we'd write would be quite fragile.
For example, if we change the column name in our database from FirstName to First_Name, our code would still compile just fine, but would throw an exception when we try to run it. Moreover, the data in the database was stored as scalar values organized in columns and rows in a table, but our destination was an object or object graph. As you can see, this way of accessing the data has a number of issues.

First of all, there is a type mismatch between RDBMS column types and .NET types. Second, there is a mismatch between storage, which is a collection of scalar values, and destination, which is an object with properties. To further complicate the situation, our person object could also have a complex property that contains a list of phone numbers, which would be represented by a completely different table. These problems are collectively referred to as **impedance mismatch** between object-oriented programming and relational databases.

The set of tools called **ORM** came about to solve this mismatch problem. An ORM tool represents data stored in database tables as objects, native to a programming language, such as .NET languages, C#, and VB.NET. ORM tools have many advantages over the traditional code, such as ADO.NET code that we mentioned. They expose the data using native .NET types. They expose related data using simple .NET properties. They provide compile time checking. They solve the problem with typos. Developers do not have to use SQL, a different language. Instead in the .NET world, developers use **Language INtegrated Query (LINQ)** to query the data. LINQ is simply part of C# and VB.NET languages. We will cover the basics of LINQ in subsequent chapters. By the same token, programmers use an ORM tool's API to persist data to the database. Finally, as we will see later, you will write less code. Less code means fewer bugs, right?

**A brief history of Entity Framework**

Over the years, there have been many ORM tools entering the market; some commercial, others open source. Microsoft developed its own tools. First one was LINQ to SQL, which was built on .NET 3.5. This ORM only worked with SQL Server and SQL Server Compact. **Entity Framework**, which first shipped in 2008, was the second attempt. It had a number of advantages over LINQ to SQL. First of all, it had provider architecture, thus was open to working with all relational database engines, not just SQL Server, given that a provider was written for the engine in question. All major RDBMSes have Entity Framework providers at this point in time.
Introducing Entity Framework

Entity Framework went through a few revisions. In the first version, only Database First approach was supported. What this meant was that you would point the designer to an existing database. As a result, code was generated that would contain a database and table abstractions. In addition to the code, an EDMX file was also created. This XML file contained Entity Data Model. It consisted of three models: logical, storage, and mapping. The logical, sometimes called conceptual, model is the one you will code against in C# or VB.NET. Storage model describes how data is stored in a database. The mapping model, as the name implies, provides the mapping between logical and storage models. If you were to change anything in the database, you would need to refresh the generated model. The C# or VB.NET code is also generated again. The mapping model has a class based on ObjectContext that has collection properties for each table in the database. Each collection is a generic collection, where collection item type is inherited from a base class in Entity Framework. Each class has properties that correspond to columns in the matching table.

In the second revision, version 4, the Model-First approach was supported as well. With this approach, you can use design surface to create entities, and then the designer would produce the SQL script to generate the database. With this approach, the EDMX file was still created, and the final result was the same as with the Database First approach. Developers had access to the same set of classes to give them the ability to persist and query data.

Finally, the Entity Framework Code-First approach was shipped in version 4.1. This approach eliminated the need for the EDMX file. It also eliminated the dependency on Entity Framework base classes that each entity in the model inherited from. As a result, the code became more testable. This approach also eliminated the need for the designer. You could just type your classes, and they would automatically be mapped to tables in the database. There have been subsequent Entity Framework Code-First releases after the initial 4.1 version.

The capabilities of Entity Framework

Entity Framework can do a lot for us as Microsoft developers. First of all, it is capable of exposing the database as a set of objects. It does so by utilizing a couple of key classes. First and foremost, you need to be aware of DbContext. This class is at the heart of Entity Framework Code-First. At a high level, it is a database abstraction. Databases consist of tables, each consisting of rows and columns. DbContext in turn has generic collection properties; each of which can be typed as DbSet<TRowType>, corresponding to each table. Each object within the collection, referred to as an entity, represents a row in the corresponding table. Columns are defined by properties of the TRowType class that is specified as a generic argument of each collection.
Once this structure is laid out, you are capable of querying the underlying database by using LINQ queries. If you add a brand new instance of the TRowType class to its parent collection and then save the changes using the DbContext API, this new object will become a row in the corresponding table, where each property value of that object will become a column value in the target row. On top of this, Entity Framework has capabilities to represent other database artifacts, such as procedures and functions. You will be able to query the data using functions, just like tables using LINQ again. The question of evolving the database structure is an important one. In most cases, you will need to add columns and tables, as your application changes. Entity Framework addresses this need via the Migrations feature. This ability will allow you to alter the database structure through C# code. In addition to adding and deleting tables and columns, you will be able to add indexes. Migrations allow developers to evolve a schema without data loss. As you can see, Entity Framework exposes everything you need to access the data in your C# or VB.NET code without wiring SQL and treats your database as another part of your overall application code. You can check migrations code into source control, since it is also C# code!

The Entity Framework architecture

Entity Framework is built on the provider architecture. When a developer creates a LINQ query using C# or VB.NET, the framework engine in conjunction with a provider converts it into an actual SQL statement that is sent to the database. Any given provider is the link between Entity Framework and a specific RDBMS that this provider is written for. In this book, we will concentrate on the Code-First approach, but this architecture is used in the Database First approach as well. Once the provider executes the final SQL command, its results are materialized into .NET objects by Entity Framework. Data reader is used for this purpose. It is important to understand that Entity Framework is still built on top of ADO.NET, thus it is uses concepts such as connection, command, and data reader. When it comes to data persistence, in other words; insert, update, and delete functionalities, the flow is as follows: In the case of inserts, a developer adds an instance of an entity class to the context. Similarly, an entity previously added to the context can be flagged as changed or deleted, causing the update or delete SQL statement to be executed against the database, respectively. Entity Framework examines the state of each object in its context, using the provider again to create an RDBMS-specific insert, update, or delete command.
Self-test questions
Q1. Which of these problems does an ORM tool solve?
   1. Types in RDBMS and .NET framework are the same
   2. Impedance mismatch between RDBMS and object-orientated programming
   3. Learning SQL is hard
Q2. Developers must write SQL queries to work with Entity Framework. True or false?
Q3. What is the name of the technology that Entity framework uses to apply structural changes to the target database?
   1. Updates
   2. Conversions
   3. Migrations
Q4. Which is the key class that represents database abstraction with the Entity Framework Code-First approach?
   1. DbContext
   2. ObjectContext
   3. DataContext
Q5. Entity Framework can only work with Microsoft databases, such as SQL Server. True or false?

Summary
In this chapter, we took a look at how data is stored in RDBMS systems. We saw the shortcomings of using embedded SQL to access the data. We understood what ORM tools are all about and what problems they solve. We examined the history behind Entity Framework. We saw the capabilities of Entity Framework. Finally, we had a brief excursion into the Entity Framework architecture.

In the next chapter, we will actually build our first application based on Entity Framework Code-First.
Where to buy this book
You can buy Code-First Development with Entity Framework from the Packt Publishing website.
Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.
Click here for ordering and shipping details.