Chapter title (H1 – Chapter)

Enter your chapter introduction here. It shouldn’t exceed much more than 300 words. Introductions should do the following (P – Regular): Introduce the topic or topic areas we’re going to cover; Tell the reader exactly what they’ll practically be doing, achieving, and learning in the upcoming lessons or activities; tell the reader why these lessons are useful. What will readers be able to do by the end of the chapter? This is your opportunity to outline a value proposition to the customer.

Add the line, “In this chapter we’re going to cover the following main topics:” Then, add a bullet list of your main chapter headers.  Your main headers should denote the main topics or tasks covered in the chapter. The purpose of this bullet list is to allow readers to easily navigate to certain sections.

* Recipe 1 (L – Bullets)
* Recipe 2
* Recipe 3

Technical requirements (H1 – Section)

In (P – Regular) this section, add the technical requirements for your chapter. List the technologies and installations required. You’ll also need to provide the Github URL, for example, <https://github.com/PacktPublishing/Getting-Started-with-TensorFlow> (P – URL), for the code in the chapter. Create a Github folder named, "chX", where X is the chapter number. For example, ch1.

Recipe 1 (H1 – Section)

The (P – Regular) language you use in your header titles should always indicate what the reader is going to do or learn in the following section. This will nearly always mean including a verb, preferably in the present participle or gerund ‘-ing’ form, for example: creating x, building y, implementing z (creating an environment, building a stack, implementing software design principles).

The opening paragraphs or the opening few sentences of each section should discuss what the reader is about to do/achieve. Your opening should also mention, if it isn’t implicit, why the lessons are applicable and useful. If the section involves completing a task, you might need to discuss the method.

Getting ready (H2 – Heading)

We will list recipe-specific requirements in this section

How to do it (H2 – Heading)

Here we list the steps required to be performed for the recipe

How it works (H2 – Heading)

The explanation of steps comes here.

There’s more (H2 – Heading)

Further application of the recio

See also (H2 – Heading)

List of related recipes that readers might be interested in learning

Recipe 2 – (H1 – Section)

Repeat Recipe 1 format

What is an ORM?

This book is about Object-Relational Mappers (ORMs in short) and their younger siblings, Micro ORMs, specifically, ORMs and Micro ORMs built with, and to be used with, the .NET framework. An ORM is a framework or library that lets you query and manipulate data coming from a relational database using an object-oriented (OO) paradigm. Or, put in a different way, an ORM helps us overcome the object-relational impedance mismatch, which happens because, of course, a relational database and an object-oriented language are totally different beasts.

When we think about relational databases, we normally think of:

* Databases
* Schemas
* Tables
* Views
* Columns
* Primary keys
* References to other tables (and foreign keys)
* Constraints and checks
* Records
* Queries

Whereas when we think about object-oriented programming (OOP) we have instead:

* Classes (and the namespaces they live in)
* Inheritance or polymorphism (base classes, inherited classes)
* Fields (and / or properties)
* Instances of classes
* Instance validation
* Methods

As you can see, there is some similarity between the two concepts:

* A class can be mapped to a table, although most relational databases don’t support inheritance
* A table or view record can be mapped to a class instance
* Table or view columns can be mapped to class fields
* A class can have a field that provides its identity, meaning, its value is unique for all instances of that class, in pretty much the same way as a table’s primary key
* A class can have fields of another class’ type, just like foreign key columns that connect two tables
* A class can be validated to check if it is in a consistent state, like a record can, even though a record only exists if its columns met all the constraints and checks
* Queries can be mapped to class methods

ORMs take care of this mapping, allowing us to only care about classes, their lifetime and properties. Retrieving records from the database and persisting them back is just a detail, something that ORMs do for us automagically. Well, almost, this is not exactly true and forgetting that we are working with a database can lead to problems and unexpected situations.

Also, modern ORMs have functionality that help us

Technical requirements

In order to follow the recipes described in this book, you will need to download the current versions of NHibernate, Entity Framework Core and Dapper.NET. All of these are available for free in both compiled (ready to use) or source code (requiring compilation) formats. Finally, you will need Visual Studio 2019, any edition, and .NET 5 installed on your machine.

What is NHibernate

NHibernate is a port for the .NET framework of the venerable Hibernate, an open-source project written in Java that is the *de facto* standard for persistence of objects into relational databases. This .NET project was started 15 years ago and is maintained by a small team of highly dedicated enthusiasts who work on their free time. Because of its age, it has many features and supports a vast number of relational databases out of the box, but it also has a steeper learning curve. In recent years, its popularity has somewhat decreased, mostly because of the arrival of Entity Framework Core.

NHibernate source code is available at <https://gihub.com/nhibernate/nhibernate-core> and its NuGet package is [https://www.nuget.org/packages/NHibernate](https://www.nuget.org/packages/NHibernate/).

What is Entity Framework Core

Entity Framework Core is Microsoft’s own ORM. It began as Entity Framework back in 2008 with a very limited set of features, and, with the announcement of the .NET Core, it has since split in two, Entity Framework “classic”, still targeting the Microsoft .NET “full” framework and Entity Framework Core, which is built on .NET Core. Microsoft has announced that it will deprecate Entity Framework “classic” and that no more major versions will be released. Entity Framework Core, as .NET Core itself, is now open-source (managed by Microsoft) and has gained significant usage because of its easiness to use and because of the features that have been introduced. Still behind NHibernate in some ways, it is quickly getting close.

Entity Framework Core is made available in binary format from NuGet (many packages, the root one being <https://www.nuget.org/packages/Microsoft.EntityFrameworkCore>) and in source code from <https://github.com/dotnet/efcore>.

What is Dapper.NET

Dapper is a micro-ORM that is also open-source and maintained by StackOverflow owners. It is a very thin layer around ADO.NET, .NET’s own database access API, and unlike its more robust siblings, it does not attempt to add any significant services or features, but instead it embraces SQL and the classic way of accessing databases in .NET.

Dapper’s NuGet package is <https://www.nuget.org/packages/Dapper> and the source code can be found at <https://github.com/StackExchange/Dapper>.

ORM Services

Modern ORMs feature several services that greatly help our lives as developers. Not all ORMs offer the same features, but some of them are quite common. I will briefly mention them here.

TODO