- For large enterprise applications, it is recommended that you use an RDBMS considering the large amount of data throughout involved.

Although many applications may use text files, XML files, or their own custom storage engines, in this chapter we will focus on the architectural aspect of designing and using a relational database for a web application.

Database Architecture and Design

Database design can also be considered as a part of the architecture of the application that uses an external database to persist data. A thoughtfully-designed database that is optimally normalized (normalization is a concept that we will learn later in this chapter) complements the system's architecture. In previous chapters, we learnt that each application demands its own unique application architecture. The same goes for database design too, where we design the schema to suit an application's unique requirements. What worked in one application's database design might not work in another.

Before going ahead with the actual database architecture and design, we need to create a plan for the database.

Database Plan

Creating a database plan is the first step in creating a database. The objective of the plan is to list the core requirements for the database as well as the scope of the database being designed. This plan will be useful in simplifying the process and mitigating any risks involved with the actual implementation of the database.

Because every application's data requirements are unique we need different plans to suit individual scenarios. Some applications might have multiple users accessing data, and this number may grow over time. Some applications might have few users, but the amount of data could be huge, and the application may have complex rules controlling that data. A proper database plan will address a project's requirements and will also serve as a functional specification for the database after it has been implemented. Therefore, the planning process will vary according to the nature and complexity of the project.

The following are the core steps in creating a plan for your database:

- Understanding the project's specific requirements
- Listing the major entities
- Indentifying the relationships between these entities

- Identifying the relationships between objects
- Addressing the scope of the application in terms of size, complexity and future scalability

Based on the project requirements, we have to create a model of the database. This model can also be called the design of the database. On a broader level, database design has two parts—logical design and physical design.

Logical Design

Logical design refers to establishing the relationship between the different system entities. For example, an employee can have multiple roles. So using a logical design, we can create relationships between the Employee entity and the Role entity.

Most projects start with the identification of the major entities in the system, and then progress to identifying relationships between these entities. Entity Relationship diagrams, or ER diagrams, which we learnt about in Chapter 3, are perfect for depicting such relationships. Once an ER diagram (which will also be the logical model of the database) has been defined, we can work on the object model design, called the **domain model**. In the domain model, we define different system entities (or objects), and define the relationships between them (such as association, aggregation, and so on). We have already created a domain model for our Order Management System (OMS) in Chapter 3.

In Chapter 3, we created a domain model and created our classes based on the domain model. This is known as Domain-Driven Design. But some projects may be heavily data-centric, with little or no business logic. Then the domain classes would not have much behavior in them. Such classes would then merely act as data containers. So in these cases, we can go for a Data-Driven approach, where we bypass the creation of a Domain model and start directly with the database entities and code using those classes. The Domain-Driven approach may take more time than the Data-Driven design, but is recommended because it helps in the long-term maintenance of the project code base by keeping the code more organized and flexible.