As promised in the introduction of this book, this final chapter discusses some object-relational features of the Oracle DBMS. For a proper understanding and appreciation of object-relational database features in general, you should consider those features in the context of an object-oriented development environment. Because this book is devoted to Oracle SQL, this chapter focuses on the consequences of these object-relational features for the SQL language.

The first step in setting up an object-relational environment is the definition of the appropriate collection of object types and methods. Once you have defined your object types, you can use them to create object tables, thus creating a truly object-relational environment. You can also use object views to create an object-relational layer on top of standard relational environments. This chapter mainly uses object types as a starting point for creating user-defined datatypes, and then using those datatypes in relational table structures.

Along with “regular” user-defined datatypes, there are two special user-defined datatypes, also referred to as collection types because they are multivalued: variable arrays and nested tables. The first four sections of this chapter cover collection types and user-defined datatypes.

Section 12.5 introduces the ANSI/ISO standard multiset operators, which allow you to perform various sophisticated operations with nested tables. Note that the PL/SQL language normally plays an important role in creating an object-relational environment. PL/SQL is the programming language you need in the definition phase of such an environment. Because PL/SQL is not covered in this book, we assume some basic knowledge of this language.

---

**Note** Instead of PL/SQL, you can also use the Java language to create an object-relational environment.

---

### 12.1 More Datatypes

So far in this book, we have used only the standard, built-in datatypes supported by Oracle, such as `NUMBER, BINARY_FLOAT, BINARY_DOUBLE, DATE, TIMESTAMP [WITH [LOCAL] TIMEZONE], INTERVAL, [N]CHAR, and [N]VARCHAR2`. This means that we haven’t discussed the following two Oracle datatype categories:

- **Collection datatypes**: These are variable arrays (`varrays`) and `nested tables`. You are probably familiar with the concept of arrays from other programming languages, and nested tables are tables within a table.

- **User-defined datatypes**: These allow you (as the name indicates) to define your own complex datatypes.
Collection Datatypes

Collection datatypes are a special case of user-defined datatypes. Collection datatypes support attributes that can have multiple values. For example, you can store a list of phone numbers for each employee in a single column, or you can add a set of errata entries to every row in the COURSES table.

The first example (adding a list of phone numbers) is an obvious candidate for using a varray, because, in general, you know the maximum length of such a list of phone numbers in advance. Also, you probably want to assign some meaning to the order of the phone numbers in the list (office extension, home, mobile, fax, and so on).

It is probably better to implement the second example (maintaining course errata) with a nested table, because you don’t have an idea beforehand about how many errata entries to expect. Also, the physical order of those errata is irrelevant, as long as you store enough errata attributes.

As you will see soon, you cannot create nested tables without using user-defined datatypes.

As a user-defined datatype, you might, for example, create an ADDRESS type, with STREET, NUMBER, POSTALCODE, and CITY components. You can create arrays of user-defined datatypes. For example, you could use the ADDRESS type to add an array of addresses to the OFFERINGS table. That would allow you to store multiple alternative location addresses for course offerings. If you want to store only a single location address, you obviously don’t need an array—a regular user-defined address type would be sufficient.

Methods

You can add methods to user-defined datatypes. Methods are operations specifically developed to work with your user-defined datatypes; for example, to specify how you want to compare two address type values, or how you want to sort address values.

Methods add a lot of semantic power to your user-defined datatypes. Unfortunately we can’t spend much time on methods in this book, because you need a great deal of PL/SQL programming to create methods. If you want to see some method examples, check out the CUSTOMERS table of the OE schema, one of the standard sample schemas that ships with the Oracle software.

As you will see in the next section, as soon as you create a user-defined datatype in Oracle, you implicitly get one method “for free”—a method with the same name as the datatype itself. That method is the constructor method, which allows you to create occurrences of the datatype.