Chapter 1

PL/SQL and Your Database

In This Chapter

- Getting to know relational databases
- Understanding database terminology
- Finding out about Oracle
- Using SQL and PL/SQL
- Discovering what PL/SQL is good for

PL/SQL is an extension to the industry-standard SQL language. Oracle Corporation developed PL/SQL and released the first version in 1991. PL/SQL is an easy-to-use procedural language that interacts seamlessly with the Oracle database. Server-side PL/SQL is part of the Oracle database and needs no explicit installation or licensing.

This chapter introduces you to PL/SQL and provides some basics about relational databases.

Knowing Just Enough about Relational Databases

Building a system in Oracle or some other relational database product does not automatically make it a relational database. Similarly, you can design a perfectly good relational database and implement it in something other than a relational database product. We discuss two important areas:

- What do people mean by *relational database?*
- What is the Oracle relational database product?

What makes a database "relational"?

When a database is described as *relational,* it has been designed to conform (at least mostly) to a set of practices called the *rules of normalization.* A normalized database is one that follows the rules of normalization.

For example, in an organization, you have employees who work in specific departments. Each employee and department has a number and a name. You could organize this information as shown in Table 1-1.

Table 1-1 Sample Employee Information

EmpNo	*Ename*	*DeptNo*	*DeptName*
101	Abigail	10	Marketing
102	Bob	20	Purchasing
103	Carolyn	10	Marketing
104	Doug	20	Purchasing
105	Evelyn	10	Marketing

If you structure your data this way and make certain changes to it, you'll have problems. For example, deleting all the employees in the Purchasing department will eliminate the department itself. If you change the name of the Marketing department to "Advertising," you would need to change the record of each employee in that department.

Using the principles of relational databases, the Employee and Department data can be restructured into two separate tables (`DEPT` and `EMP`), as shown in Tables 1-2 and 1-3.

Table 1-2 A Sample Relational DEPT Table

DeptNo	*DeptName*
10	Marketing
20	Purchasing

Table 1-3 A Sample Relational EMP Table

EmpNo	*EName*	*DeptNo*
101	Abigail	10
102	Bob	20
103	Carolyn	10
104	Doug	20
105	Evelyn	10

By using this structure, you can examine the `EMP` table to find out that Doug works in department 20. Then you can check the `DEPT` table to find out that department 20 is Purchasing. You might think that Table 1-1 looks more efficient. However, retrieving the information you need in a number of different ways is much easier with the two-table structure. Joining the information in the two tables for more efficient retrieval is exactly the problem that relational databases were designed to solve.

When the tables are implemented in the database, the information in the two tables is linked by using special columns called *foreign keys.* In the example, the `DeptNo` column is the foreign key linking the Department and Employee tables.

Tables 1-4 and 1-5 show another common database structure, namely a purchase order (`PURCH_ORDER` table) for an item and the information details associated with the purchase order (`PURCH_ORDER_DTL` table).

Table 1-4 A Sample Relational PURCH_ORDER Table

PO_Nbr	*Date*
450	12/10/2006
451	2/26/2006
452	3/17/2006
453	6/5/2006

Table 1-5 A Sample Relational PURCH_ORDER_DTL Table

PO_Nbr	*Line_Nbr*	*Item*	*Qty*	*Price*
450	1	Hammer	1	$10.00
451	1	Screwdriver	1	$8.00
451	2	Pliers	2	$6.50
451	3	Wrench	1	$7.00
452	1	Wrench	3	$7.00
452	2	Hammer	1	$10.00
453	1	Pliers	1	$6.50

A purchase order can include many items. Table 1-5 shows that Purchase Order 451 includes three separate items. The link (foreign key) between the tables is the Purchase Order Number.

Understanding basic database terminology

A database consists of tables and columns, as we describe in the preceding section. There are some other terms you need to know in order to understand how databases work. A database is built in two stages. First you create a *logical data model* to lay out the design of the database and how the data will be organized. Then you implement the database according to the *physical data model,* which sets up the actual tables and columns. Different terminology applies to the elements of the logical and physical designs. In addition, relational database designers use different words from object-oriented (OO) database designers to describe the database elements. Table 1-6 shows the words used in each of these cases.

Table 1-6 Database Design Terminology

Logical/Relational	*Logical/Object-Oriented*	*Physical Implementation*
Entity	Class	Table
Attribute	Attribute	Column
Instance	Object	Row

The definitions of the words in Table 1-6 are as follows:

- **Entity:** An entity corresponds to something in the real world that is of interest and that you want to store information about. Examples of entities include things such as departments within an organization, employees, or sales. Each specific department or employee is considered an *instance* of that entity. For example, in Table 1-3, `Doug` is an instance of the entity `Employee`. (In the OO world, `Doug` would be an object in the `Employee` class.)
- **Attribute:** This word is used in both relational and OO databases to represent information about an entity instance or an object that will be tracked. An example of an attribute might be the birth date or Social Security number of an employee.
- **Entities (classes), their attributes, and instances (objects):** These are implemented in the database as tables, columns, and rows respectively.

One additional important concept to understand when dealing with relational databases is the primary key. A *primary key* uniquely identifies a specific instance of an entity. No two instances of an entity can have the same primary key. The values of all parts of the primary key must never be null. The most common types of primary keys in relational databases are ID numbers. For example, in Table 1-3, the `EmpID` can be the primary key. Sometimes more than one attribute (or sets of attributes) can be used as a primary key. These attributes are called *candidate keys,* one set of which must be designated as the primary key.

Introducing database normalization

A database is considered *normalized* when it follows the rules of normalization. Database normalization is useful for several reasons:

- It helps to build a structure that is logical and easy to maintain.
- Normalized databases are the industry standard. Other database professionals will find it easier to work with your database if it is normalized.
- Retrieving data will be easier. This is actually the formal reason to normalize. Graduate students in database theory courses often have to prove a theorem that roughly states, "If your database is normalized, you can be sure that any set of information you want to retrieve from your database can be done by using SQL."

You frequently need very complex procedural code to extract information from a non-normalized database. The rules of normalization will help you to design databases that are easy to build systems with.

Although a detailed discussion of normalization is beyond the scope of this book, there are three basic rules of normalization that every database professional should have memorized. Not so coincidentally, we tell you about them in the following three sections.

First Normal Form (1NF)

First Normal Form means that the database doesn't contain any repeating attributes. Using the Purchase Order example from Tables 1-4 and 1-5, the same data could be structured as shown in Table 1-7.

Table 1-7 PURCH_ORDER Table (1NF Violation)

PO_NBR	DATE	ITEM 1	QTY1	PRICE1	ITEM2	QTY2	PRICE2
450	12-10-06	Hammer	1	$10.00			
451	02-26-06	Screwdriver	1	$8.00	Pliers	2	$6.50
452	03-17-06	Wrench	3	$7.00	Hammer	2	$10.00
453	06-05-06	Pliers	1	$6.50			

Although this table looks okay, what if a third item were associated with PO 451? Using the structure shown in Table 1-7, you can order only two items. The only way to order more than two items is to add additional columns, but then to find out how many times an item was ordered, you'd need to look in all the item columns. Table 1-7 violates First Normal Form.

You can build a good database that doesn't adhere to First Normal Form by using more complex collections such as VARRAYs and nested tables (which we discuss in Chapter 11).

Second Normal Form (2NF)

Violations of *Second Normal Form* occur when the table contains attributes that depend on a portion of the primary key.

To talk about Second Normal Form, you should know what we mean by an attribute being *dependent* on another attribute. Say attribute X is dependent upon attribute Y. Then if you know the value of attribute X, you have enough information to find the value of attribute Y. Logically, attribute Y can have only one value. For example, from the information in Table 1-1, if you know the Employee Number (`EmpNo`), you also know the employee's name, which department number he or she works in, and the number of that department. In this case, the `EmpNo` is the primary key. However, knowing the department number and department name doesn't tell you a specific employee's name or number. You can't use the department number/name combination as the primary key. You can't even use the name (`Ename`) as the primary key because a large organization might have more than one "John Smith" working there.

Second Normal Form violations can exist only when you have a multi-column primary key, such as the purchase order and the purchase order detail structure, as shown in Tables 1-8 and 1-9.

Table 1-8 PURCH_ORDER Table

PO_NBR	*DATE*	*Vendor*
450	12-10-06	ABC Co.
451	02-26-06	XYZ Inc.
452	03-17-06	XYZ Inc.
453	06-05-06	ABC Co.

Table 1-9 PURCH_ORDER_DETAIL Table (2NF Violation)

PO_NBR	*LINE*	*DATE*	*ITEM*	*QTY*	*PRICE*
450	1	12-10-06	Hammer	1	$10.00
451	1	02-26-06	Screwdriver	1	$8.00
451	2	02-26-06	Pliers	2	$6.50
452	1	03-17-06	Wrench	3	$7.00
452	2	03-17-06	Hammer	2	$10.00
453	1	06-05-06	Pliers	1	$6.50

In this structure, the `PURCH_ORDER_DETAIL` table uses both `PO_NBR` and `LINE` for the primary key. But `DATE` is dependent only on the `PO_NBR` (when you know the `PO_NBR`, you know the date that each item was ordered), so that column violates Second Normal Form.

Third Normal Form (3NF)

Third Normal Form violations occur when a *transitive dependency* exists. This means that an attribute ID is dependent on another attribute that isn't part of either a primary or candidate key. These are serious violations indicating errors in the database design that must be detected and corrected. Table 1-1 shows an example of Third Normal Form violation in a badly designed database. The `DeptName` column is dependent only on the `DeptNo` column (that is, if you know the department number, you know the name of the department). The `EmpNo` is the obvious primary key, so the existence of `DeptName` column violates Third Normal Form.

All attributes in entities (columns in tables) must be dependent upon the primary key or one of the candidate keys and not on other attributes.

For more information about normalization, look at books about database theory such as *Beginning Database Design,* by Gavin Powell (Wiley) and *A First Course in Database Systems,* by Jeffrey D. Ullman and Jennifer Widom (Prentice Hall), or numerous works by Chris J. Date.

What is a DBMS?

After you've designed a relational database, you need to implement it. The easiest way to do this is by using a product that's specifically designed for this purpose. Products that perform these operations are called Relational Database Management Systems (usually abbreviated to RDBMS or just DBMS). They allow you to easily create relational databases by defining and creating tables and then populating them with data. In addition, you could be provided with a special tool to modify and manipulate the data and write reports and applications to interact with the data.

DBMSs also handle all sorts of other important functions. They allow many people to access the database at the same time without interfering with one another or corrupting the data. They also make it easy to create backups in case of problems such as a power failure or other disasters.

A number of positions in Information Technology involve interaction with a DBMS:

- **Database designer:** This person analyzes the requirements for the system and designs an appropriate database structure to house the data.
- **Database administrator (DBA):** This person installs the DBMS, monitors it, and physically manages its operations.
- **Database application developer:** This person writes the code that resides within the DBMS and directly interacts with the database.
- **User interface (UI) application developer:** This person writes the code for the user interface, which enables users to communicate with the database.

Many other people, including project managers, software testers, and documentation specialists, also work with database systems. This book focuses on the skills required to be a database application developer.

The Scoop on SQL and PL/SQL

As a database application developer, you interact with the Oracle DBMS by using the programming languages Structured Query Language (SQL,

pronounced *sequel*) and Programming Language/Structured Query Language (PL/SQL, pronounced either P-L-S-Q-L or P-L-*sequel*). In the following sections, we introduce how SQL and PL/SQL work together and how they are different. We also introduce what's new in the current versions.

The purpose of SQL and PL/SQL

SQL is the industry standard language for manipulating DBMS objects. Using SQL, you can create, modify, or delete database objects. This part of SQL is called Data Definition Language (DDL). You can also use SQL to insert, update, delete, or query data in these objects. This part of SQL is called Data Manipulation Language (DML).

Oracle's implementation of SQL isn't exactly industry standard. Virtually every DBMS (Oracle included) has invented items that are not part of the standard specification. For example, Oracle includes sequences and support for recursive queries that aren't supported in other DBMS products.

Oracle is more than a database

The Oracle environment doesn't consist solely of the DBMS. The Oracle environment itself is enormous and complex, and the large number of products that Oracle sells is a reflection of that. So how does the DBMS fit into the bigger picture? Here's a quick overview of the main categories of Oracle products:

- **Oracle DBMS:** This database management system runs on a variety of computers and operating systems. As we write this book, it's often considered to be the largest, fastest, most powerful, and fully featured database product on the market. The Oracle DBMS is the industry standard for big companies that need to store and manipulate large volumes of data. Oracle also provides versions of the DBMS to support small and medium-sized companies.
- **Application development software:** Oracle has many application development products. The current main product is JDeveloper, a Java-based programming environment.
- **Oracle Application Server (OAS):** Web-based applications typically run on a dedicated computer. Oracle's version of this is called OAS.
- **Oracle Applications:** Oracle has created or acquired a number of enterprise-wide applications that work with the Oracle DBMS and help Accounting, Manufacturing, and Human Resources departments to perform their day-to-day functions more efficiently.

Oracle Corporation also includes consulting (Oracle Consulting) and education (Oracle University) divisions to round out its offering of products and services.

Getting to know SQL in an Oracle environment allows you to work in almost any DBMS environment, such as SQLServer or MySQL, but you'll encounter some differences in the DBMS environments. You should probably know SQL before trying to use PL/SQL. This book assumes that you already know SQL. If you haven't mastered SQL, take a good long look at *SQL For Dummies,* 5th Edition, by Allen G. Taylor (Wiley), before you dive into this book.

PL/SQL is unique to Oracle. It isn't industry standard. No other product uses it. Being able to use PL/SQL will help you work only within the Oracle database environment, but if you're familiar with any other programming language, you'll find that PL/SQL follows the same basic rules.

PL/SQL is similar to other non-object-oriented procedural programming languages, such as C or Pascal. Its intellectual roots go back to a programming language called Ada.

What makes PL/SQL unique is its tight integration with SQL. It is easier and more natural to embed SQL in PL/SQL than to do so in any other programming language. This makes PL/SQL ideal for writing large, complex programs that must interact with an Oracle database.

The difference between SQL and PL/SQL

SQL and PL/SQL are completely different languages. SQL is a limited language that allows you to directly interact with the database. You can manipulate objects (DDL) and data (DML) with SQL, but SQL doesn't include all the things that normal programming languages have, such as loops and `IF...THEN` statements.

That is what PL/SQL is for. PL/SQL is a normal programming language that includes all the features of most other programming languages. But it has one thing that other programming languages don't have, namely the easy ability to integrate with SQL.

What's new in Oracle SQL and PL/SQL?

Oracle SQL and PL/SQL are evolving languages that constitute the backbone of applications written for the Oracle environment. Every version of the Oracle database expands the features of these languages. The production version of Oracle 10*g* Release 2 has recently been released. As with previous versions, this release offers lots of new things, including the following:

- PL/SQL will probably run faster in the 10*g* version than it did in previous versions. You don't have to do anything extra to benefit from that improvement. Oracle has made PL/SQL code run faster without requiring any additional work on the part of the programmer.

- In SQL, many new commands allow you to retrieve information more easily than before. Information about these commands is beyond the scope of this book, but make sure you have a good Oracle SQL book, such as *Oracle Database 10g: The Complete Reference,* by Kevin Loney (McGraw-Hill), as a source for all the commands.

Because every release brings new capabilities, keeping up with the new features in Oracle is important. Many developers don't keep up with new features because "all the old features will still work," but those developers miss out on the great new features included in each version. If you do a search for "new features in PL/SQL" or "new features in Oracle SQL" in Google or your favorite search engine, you'll always find many articles and resources to show you the latest additions to these programming languages.

What Is PL/SQL Good For?

PL/SQL is the language to use when writing code that resides in the database. In the following sections, we introduce different situations in which you'll find PL/SQL useful.

Using database triggers

A *trigger* is an event within the DBMS that can cause some code to execute automatically. There are four types of database triggers:

- **Table-level triggers** can initiate activity before or after an `INSERT`, `UPDATE`, or `DELETE` event. These are most commonly used to track history information and database changes, to keep redundant data synchronized, or to enhance security by preventing certain operations from occurring. See Chapter 3 for more information about table-level triggers.
- **View-level triggers** are very useful. A *view* is a stored SQL statement that developers can query as if it were a database table itself. By placing `INSTEAD OF` triggers on a view, the `INSERT`, `MODIFY`, and `DELETE` commands can be applied to the view regardless of its complexity, because the `INSTEAD OF` trigger defines what can be done to the view. See Chapter 3 for more information about view-level triggers.
- **Database-level triggers** can be activated at startup and shutdown. For example, when the database starts up you might want to test the availability of other databases or Web services. Before a database shutdown, you might want to notify other databases and Web services that the database is going offline.
- **Session-level triggers** can be used to store specific information. For example, when a user logs on or off, you might want to execute code

that contains the user's preferences and loads them into memory for rapid access. When the session closes, a trigger can save the preferences for future use.

Database and session-level triggers are usually handled by DBAs, and further discussion of their use is beyond the scope of this book.

Scripting with speed

When writing code, the ability to type a portion of code and execute it without first saving it to the database is useful. Oracle provides this capability, which is supported by all PL/SQL IDEs. We discuss this capability in Chapter 2.

Keeping code server-side

The majority of PL/SQL code is stored as program units in the server. A typical application has many lines of code.

Some programmers, particularly Web-based developers working in the J2EE or .NET environments, try to write most of their code in the application server in Java (for J2EE developers) or VB.NET (for .NET developers). This isn't good practice. In a database application, much of the logic is devoted to retrieving and updating information. If the code to accomplish this task resides in an application server, it must send a request to the database over a network. Then the database must process the request and send the information back across the network for the application to process. Because networks and computers are now very fast, you might think that this would take only fractions of a second. Although this is the case for a single request, if a very complex application requires millions or even hundreds of millions of interactions with the database, multiplying the number of interactions by even fractions of a second can lead to very poor performance.

Even relatively simple operations requiring only a few database requests can be problematic if the application is being accessed by hundreds, thousands, or tens of thousands of users simultaneously. It is much more difficult to build a database-intensive application without using server-side coding than it is to write all the code to run in an application server.

One of the arguments against writing server-side code is that the application won't be portable (can't be moved from one platform to another). However, most organizations using Oracle have been using it for a very long time (ten or more years) and aren't contemplating a switch to a different platform. Also, Web development is currently in a state of rapid flux. Organizations frequently change between .NET, J2EE, and other environments for their Web-based application development.

Both the .NET and J2EE environments are in flux, as well. In the J2EE environment, the industry standard for Web development a year or so ago was to create JavaServer pages (JSPs). Currently, the industry standard is to work in the JSP/Struts environment. In the next year or so, JavaServer Faces (JSFs) will likely become the industry standard. Therefore, code written in the middle-tier runs a high risk of needing to be rewritten in the future.

Server-side code runs faster, is easier to maintain and test, and is less susceptible to change than code placed in the middle tier. Therefore, creating significant portions of an application in the database is a better approach.

There are a number of places where you can write code that your applications can use. We discuss each in turn:

- **Portions of applications:** PL/SQL program units can return a set of values (functions), or PL/SQL routines can perform database operations (procedures). These functions and procedures may be called by other functions and procedures or (in the case of functions) used in SQL statements. PL/SQL routines may be as large and complex as you need them to be. Some complex routines may contain thousands of lines of code. Entire systems may contain millions of lines of code. Chapter 3 covers the creation of functions and procedures and how to place them into packages.
- **PL/SQL code embedded in views:** Oracle allows you to embed code in database views. The code might actually be located in one of two places in the view. First, you can place correctly crafted functions returning a value in the `SELECT` portion of a SQL statement to retrieve additional information, which might or might not be part of the tables being queried. For example, you can create a view of a `Customer` table with a function that would return the amount currently owed, even if this amount involves a complex calculation and is not stored in the `Customer` table.

  You can also embed PL/SQL in `INSTEAD OF` triggers on a view. These triggers allow you to perform `INSERT`, `UPDATE`, and `DELETE` operations on complex views, with PL/SQL programmatically handling how these operations should be handled. Chapter 6 tells you about embedding code in views.
- **Batch routines:** *Batch routines* run code that processes a large number of records at the same time. Generating invoices for every customer in a system or processing payroll checks for an entire organization are examples of batch routines. These routines are usually large, complex, and database intensive. This type of routine should assuredly be written in PL/SQL.

Programming for Oracle Developer

Oracle Developer used to be the Oracle Corporation's primary application development tool. More recently, Oracle's JDeveloper has been used for Java-based applications. However, many organizations still use Oracle Developer

for internal application development — mostly development for systems that handle things like payroll.

Oracle Developer consists of two main parts:

- **Oracle Forms:** A user interface screen building tool
- **Oracle Reports:** A reporting tool

Both of these tools use PL/SQL as their programming language. The advantages to this are numerous, because the code used to create the applications is the same as that used in the database itself. Because the J2EE and .NET environments have emerged, developers must use one programming language for applications and a separate language for server-side development. Although Oracle made some efforts to make Java work within the Oracle database as PL/SQL does, the efforts weren't entirely successful.

If you're involved in a new project, the probability of using Oracle Forms is fairly low. Most new development isn't being done in Forms. However, many organizations are still using large Forms-based systems that require ongoing modifications and enhancements.

For reporting, Oracle Reports is still the primary tool for working with Oracle databases. It continues to be enhanced. Further discussion of Oracle Developer is beyond the scope of this book. For more information, see *Oracle Developer: Advanced Forms & Reports,* by Peter Koletzke and Dr. Paul Dorsey (McGraw-Hill).