CHAPTER 1

Introduction to PL/SQL

e've seen some really well-written code make some really lousy applications. Look at some of the beautifully written viruses that are out there, or some of the now-defunct software companies that turned out flashy but useless applications! Programming is more than just syntax. It is a profession where knowledge can be combined with ingenuity, communication, attitude, and discipline to build a successful career and world-class applications.

Throughout this book, we focus on more than syntax and rules. We answer the "Why would I use that?" question we all ask when shown new capabilities. Our discussions go beyond the fact that Oracle *can* do something. We show *how* and *why* it does it.

In this first chapter we set the stage for the rest of the book. The following points are discussed:

- SQL and its interaction with the relational database
- How PL/SQL uses SQL to increase capabilities
- Programming concepts, comparing procedural languages to object-oriented programming
- PL/SQL history and features
- The benefits (and drawbacks) of the language
- How to approach the remainder of this book and get the most out of this fully revised text

Introduction to Programming Languages

Java, C++, PL/SQL, and Visual Basic are some of the most popular programming languages in use today. Each one is quite different from the next, having its own unique characteristics. Even though they are distinct languages, some of them share common traits. Programming languages can be categorized according to these commonalities. The languages just listed fit into two categories: procedural and object-oriented.

Procedural languages, such as PL/SQL and Visual Basic, are linear. They *begin* at the beginning, and *end* at the end. This is a simplistic definition, but nevertheless a primary differentiator between procedural and object-oriented languages. Each statement must wait for the preceding statement to complete before it can run. For many beginning programmers, cutting their teeth on a procedural language is the best way to learn. You have a series of steps your program must perform, and that is exactly how the code works—step-by-step.

Object-oriented programming (OOP) languages such as Java and C++ are more abstract in nature. OOP languages work with structures called *objects.* For example,

instead of writing code to pull together information about a book directly from the data structures, we can create an *object* called BOOK. Each *object* has *attributes*: number of pages, price, title, etc. Attributes describe the object. *Methods* are more action oriented. They operate on the data, retrieving it or modifying it. Should you want to change the price, for example, you call a method to perform this task. This differs from a procedural language, where you would execute a series of steps to produce the same effect.

In a rare dual-category listing, PL/SQL can now be considered both procedural and object-oriented. Oracle 8 introduced objects, though in the initial releases the support for advanced features such as inheritance, type evolution, and dynamic method dispatch were not provided. With Oracle 9*i*R1, Oracle began a major push to fully support object-oriented programming with PL/SQL. As of Oracle 10*g*, most major OO features are fully supported.

NOTE
The object-oriented features mentioned here are explained in detail in Chapters 14 and 15.

Note to Beginning Programmers

Like many developers, I cut my teeth on Basic. The syntax was easily learned, yet the programming "truths" that applied to Basic applied to most other languages. I believe you will find the same true with PL/SQL.

My favorite feature of PL/SQL is not its tight integration with the database (though it is tightly integrated), advanced language concepts and capabilities (by the end of this book, you will be amazed at what can be done), or any other type of functionality it provides. My favorite feature is its structured approach to programming. For every BEGIN, there is an END. For every IF, there is an END IF.

As an instructor teaching PL/SQL to many students new to programming (not just new to PL/SQL), I know that you can learn this language. It is structured, linear, and not very forgiving. This is a good thing! You will learn structure and rules. If you do not follow the rules, you get instant feedback when trying to run your code.

```
Warning: Procedure created with compilation errors.
```

TIP
Obviously, structure does not guarantee good code. It simply makes the language easier to learn. Take caution to use good form, adopt proper naming conventions, document your actions, and practice, practice, practice. Do not allow yourself to take shortcuts that make your code inefficient and difficult to maintain. As with any language, you can write terrible code that compiles.

You may have noticed that the best programmers are not necessarily the most technically gifted. The best programmers are good communicators who have the ability to put themselves in the shoes of their users and customers. The design phase is where this is especially critical. You meet with project managers, other developers, DBAs, end users, QA engineers, and management. Each group of people has different objectives during the systems development life cycle, and each group will place different demands on you. Your attitude and ability to communicate spells the success or failure of the project and ultimately determines how far you can go in this industry.

PL/What?

So, what is PL/SQL? It is the procedural (and sometimes object-oriented) programming extension to SQL, provided by Oracle, exclusively for Oracle. If you are familiar with another programming language called Ada, you will find striking similarities in PL/SQL. The reason they are so similar is that PL/SQL grew from Ada, borrowing many of its concepts from it.

The PL in PL/SQL stands for *procedural language.* PL/SQL is a proprietary language not available outside the Oracle Database. It is a third-generation language (3GL) that provides programming constructs similar to other 3GL languages, including variable declarations, loops, error handling, etc. Historically, PL/SQL was procedural only. As discussed in the preceding section, however, PL/SQL can now be considered part of the object-oriented category of languages. Should we change the name to PL/OO/SQL?

Structured Query Language (SQL)

The SQL in PL/SQL stands for *structured query language.* We use SQL to SELECT, INSERT, UPDATE, or DELETE data. We use it to create and maintain objects and users, and to control access rights to our instances.

SQL (pronounced as *sequel* or by its letter abbreviation) is the entrance, or window, to the database. It is a fourth-generation language (4GL) that is intended to be easy to use and quick to learn. The basic SQL syntax is not the creation of Oracle. It actually grew out of the work done by Dr. E.F. Codd and IBM in the early 1970s. The American National Standards Institute (ANSI) recognizes SQL and publishes standards for the language.

Oracle supports ANSI-standard SQL but also adds its own twist in its SQL*Plus utility. Through SQL*Plus, Oracle supports additional commands and capabilities that are not part of the standard. SQL*Plus is a utility available in multiple forms:

- **Command line** From the Unix prompt or DOS prompt

- **GUI** SQL*Plus Client, SQL Worksheet, Enterprise Manager
- **Web Page** iSQL*Plus, Enterprise Manager in 10*g*

With just a client installed, we can configure a network connection to remote databases. Oracle 10*g* makes configuration even easier with a browser-based Enterprise Manager and iSQL*Plus, both configured at install time.

Relational Database Overview

SQL is the window to the database, but what is the database? A *database* in general terms is anything that stores data. Electronic databases can be as simple as a spreadsheet or word processing document.

As you might imagine, storing large amounts of data in a spreadsheet or word processing document can become overwhelming very quickly. These one-dimensional databases have no efficient way of filtering redundant data, ensuring consistent data entry, or handling information retrieval.

Oracle is a *relational database management system,* or RDBMS. Relational databases store data in tables. *Tables* are made up of columns that define the type of data that can be stored in them (character, number, etc.). A table has a minimum of one *column.* When data is placed in the table, it is stored in *rows.* This holds true for all relational database vendors (see Figure 1-1).

In Oracle, tables are owned by a user, or schema. The schema is a collection of objects, like tables, that the database user owns. It is possible to have two tables in one database that have the same name as long as they are owned by different users.

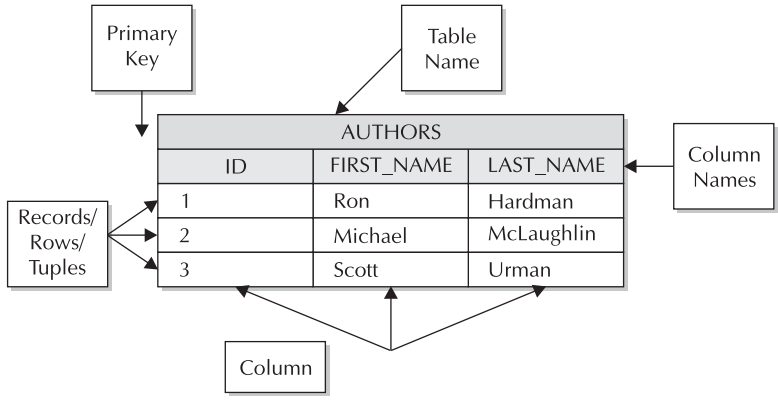


FIGURE 1-1. *Table structure*

Other vendors do not necessarily follow this approach. SQL Server, for example, applies different terminology. The SQL Server database is more like an Oracle schema, and the SQL Server *server* more resembles the Oracle database. The result is the same, however. Objects, such as tables, always have an owner.

It is possible to store all of our data in a single table, just like the spreadsheet, but that does not take advantage of Oracle's relational features. For example, a table containing data about Oracle Press books is incomplete without author information. It is possible that an author has written multiple titles. In a flat-file, or single-table, model, the author is listed multiple times. This redundancy can be avoided by splitting the data into two tables with a column that links related data together. Figure 1-2 illustrates how we can break this into two separate tables.

In Figure 1-2 there are two tables, `AUTHORS` and `BOOKS`. Author information stores the first and last names of authors one time. Each row of data is given an `ID` that is guaranteed unique and not null (null means empty, so not null means not empty).

Since we have the `AUTHORS` table, we don't have to repeat author information over and over for every title each person writes. We add a single `AUTHOR1` column in the `BOOKS` table and insert the appropriate `ID` value from the `AUTHORS` table for each title in the `BOOKS` table. Using a `FOREIGN KEY` on the `BOOKS.AUTHOR1` column, we can relate the two tables together using SQL. Let's take a look at an example:

NOTE
You may wish to use the CreateUser.sql *script located in this chapter's directory on the web site. It creates a user called plsql and grants required permissions to the user.*

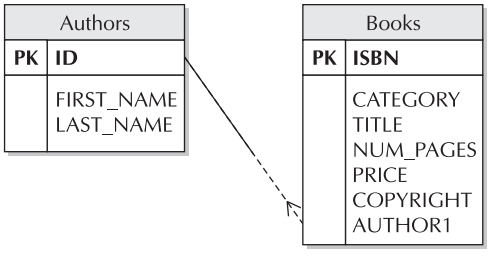


FIGURE 1-2. *ERD for Books and Authors*

```
-- Available online as part of PlsqlBlock.sql
CREATE TABLE authors (
  id         NUMBER PRIMARY KEY,
  first_name VARCHAR2(50),
  last_name  VARCHAR2(50)
);

CREATE TABLE books (
  isbn      CHAR(10) PRIMARY KEY,
  category  VARCHAR2(20),
  title     VARCHAR2(100),
  num_pages NUMBER,
  price     NUMBER,
  copyright NUMBER(4),
  author1   NUMBER CONSTRAINT books_author1
            REFERENCES authors(id)
);
```

After inserting a few records into the tables, we can perform a SELECT, joining the tables according to their relationship.

```
SELECT b.title, a.first_name, a.last_name
FROM authors a, books b
WHERE b.author1 = a.id;
```

This joins the two tables together and retrieves data just as you would have seen it had it been stored in a flat file. The differences are less redundancy, fewer opportunities for error, and greater flexibility. To add publisher information, all I would need to do is create a table called `PUBLISHER` that contains an `ID`, then add a column to the `BOOKS` table with a `FOREIGN KEY` pointing back to the `PUBLISHER.ID` column.

NOTE
For expanded coverage of SQL, refer to the online documentation at http://otn.oracle.com.

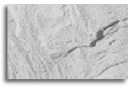
PL/SQL vs. SQL

SQL gives us complete access to our data. By complete, I mean we can get to everything. . . eventually. . . in less than ideal ways in many cases. There is no

guarantee of efficiency, and few actual programming capabilities found in most languages are possible. SQL provides no ability to

- Loop through records, manipulating them one at a time.
- Keep code secure by offering encryption, and storing code permanently on the server rather than the client.
- Handle exceptions.
- Work with variables, parameters, collections, records, arrays, objects, cursors, exceptions, BFILEs, etc.

While SQL is powerful, and SQL*Plus (Oracle's proprietary SQL interface) includes commands and built-in functions not found in the ANSI standard, SQL remains more of a method of access to the database than a programming language. PL/SQL takes over where SQL leaves off by adding the features mentioned here and more.



NOTE
Do not worry if you do not know what all of the programming features mentioned here are! That is what this book is for. They are explained in detail in later chapters.

Virtually all SQL capabilities are possible with PL/SQL. In fact, as of Oracle 9*i*R1, the PL/SQL parser is the same as the SQL parser, ensuring that commands are treated the same regardless of where they are executed. Prior to Oracle 9*i*R1, you would find some cases where a SQL statement was treated completely differently. Not so anymore.

Let's take the query of the BOOKS and AUTHORS tables that we did earlier and use it in a PL/SQL example.

```
-- Available online as part of PlsqlBlock.sql
SET SERVEROUTPUT ON
DECLARE
   v_title books.title%TYPE;
   v_first_name authors.first_name%TYPE;
   v_last_name authors.last_name%TYPE;

   CURSOR book_cur IS
      SELECT b.title, a.first_name, a.last_name
      FROM authors a, books b
      WHERE a.id = b.author1;
BEGIN
```

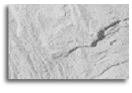
```
   DBMS_OUTPUT.ENABLE(1000000);
   OPEN book_cur;
   LOOP
      FETCH book_cur INTO v_title, v_first_name, v_last_name;
      EXIT WHEN book_cur%NOTFOUND;

      IF v_last_name = 'Hardman'
      THEN
         DBMS_OUTPUT.PUT_LINE('Ron Hardman co-authored '||v_title);
      ELSE
         DBMS_OUTPUT.PUT_LINE('Ron Hardman did not write '||v_title);
      END IF;
   END LOOP;

   CLOSE book_cur;

EXCEPTION
   WHEN OTHERS
      THEN
      DBMS_OUTPUT.PUT_LINE(SQLERRM);
END;
/
```

This example includes the select statement we used earlier, but it loops through all of the query results, determines if `'Hardman'` is the last name of the author, and formats the output accordingly. The power of SQL 4GL is combined with the features of a procedural 3GL language.



NOTE
Take note of the structure in the last block.
For every begin, there is an end.

PL/SQL vs. Java

Oracle 8*i* introduced support for Java, and Java Stored Procedures, in the database. Why not just use Java, then?

PL/SQL is, and has always been, tightly integrated with the Oracle database. Oracle continues to improve PL/SQL performance by adding integration features such as native compilation of PL/SQL code. This means that when the code is compiled, it is converted to C (the language Oracle is written in). At run time, no interpretation between PL/SQL syntax and C is required. Performance is greatly improved—up to 30 percent over interpreted mode (the default).

Another advantage of PL/SQL is that it is very compact. You can turn a SQL statement into a PL/SQL block (blocks are discussed in Chapter 3) by simply

adding a BEGIN before the statement, and an END after it. The same cannot be said for Java. The following block of code is the most basic you can create with PL/SQL:

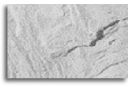
```
BEGIN
   NULL;
END;
/
```

Try it—it works. It does absolutely nothing, but it runs.

Here are some other distinctive features of PL/SQL:

- PL/SQL now shares the same parser as SQL, so there is guaranteed consistency between interfaces.
- PL/SQL can be executed from SQL.
- OO features are constantly being added to PL/SQL, removing many of the reasons to switch to Java.

This is not to say that you should *always* use PL/SQL and *never* use Java. Java includes a whole host of features not yet available in PL/SQL. Java is not a replacement for PL/SQL, though. It is simply an alternative.



NOTE
Since Java's introduction to the database, I have repeatedly heard a rumor that PL/SQL was on the way out and Java is taking over—not true.

PL/SQL History and Features

What you see as a rich feature set in the most recent releases of PL/SQL is actually 13 years (as of the time of this writing) of constant development and improvement of the language by Oracle. PL/SQL is a language developed out of need, both internal and external to Oracle. Though many of the features were created to satisfy the demands of database developers in the user community, a large number were also prompted by Oracle's need for functionality in their own application development and consulting efforts. As a developer, I find it encouraging to know that Oracle is heavily using the same technologies I rely on in my career.

It is hard to imagine the Oracle database without PL/SQL, but it was not that long ago when it was first introduced.

Version 1.x

PL/SQL 1.0 was introduced in 1991 with the 6.0 release of the data server. As you might expect with a new programming language, it was lacking in most features

you might expect from a more mature release. The Oracle development community, however, appreciated it because it gave capabilities such as IF-THEN logic that were not possible with SQL at the time.

Version 2.x

By PL/SQL version 2.3 (released with version 7.3 of the database), Oracle added support for stored procedures and functions, and added numerous built-in packages. PL/SQL was key to the success of Oracle's developer tools, and Oracle Applications relied heavily on tight integration of PL/SQL to the data server.

Version 8.0

Oracle 8.0 included support for objects. Though object support was not exactly feature-rich in this introductory release, it gave us an indication of where Oracle was taking the language. OO enhancements continue through the most recent release of 10*g*R1. One other change in 8.0 is the versioning for PL/SQL and the data server. PL/SQL began following the same version sequence as the data server it was integrated with.

Version 8.1

With Oracle 8*i,* marketing of PL/SQL features took a back seat to Java integration in the "Database for the Internet." This did not mean that there was nothing new with PL/SQL, though. One of my favorite enhancements: Native Dynamic SQL (NDS) gave us EXECUTE IMMEDIATE! I love this command—in fact, you will see it in nearly every schema creation script in the examples for this book.

Version 9.0

Oracle 9*i*R1 was a huge release for PL/SQL. The following list summarizes some of the major improvements:

- SQL and PL/SQL now share the same parser, ensuring consistency. Prior to this improvement, a statement that succeeded in the SQL*Plus window would not be guaranteed to work in PL/SQL.
- Character semantics, which allows us to define our variable or column precision in characters or bytes, was added in 9*i*R1. Unicode characters are not all created equal. They can differ in byte size. Precision in Oracle is actually in bytes, not characters! A variable declaration specifying `VARCHAR2(2)` means that the variable can hold two bytes, not two characters. Some Asian characters are up to three bytes, which means that an assignment of a single Chinese character may not fit into a variable with a precision of two. Now that is annoying!

- Support for objects now includes inheritance and type evolution. These were glaring weaknesses in PL/SQL's OO support.
- Native compilation allows PL/SQL code to be compiled as C code (Oracle is written in C), reducing time to execute, since no interpretation is required at run time.

Version 9.2

Many of the Oracle 9*i*R2 features were improvements of 9*i*R1 enhancements. Object features were improved, adding built-in functions and support for user-defined constructors. Oracle Text introduced the CTXXPATH, providing improved PL/SQL access to XML documents stored in the XMLTYPE datatype.

Version 10.0

PL/SQL 10.0 added a number of new features:

- Arguably the most important addition to 10*g*R1 PL/SQL is support for regular expressions. Regular expressions have long been a staple of Unix and Perl scripting, and they are now available with Oracle and supported in PL/SQL. The short definition: Regular expressions find, retrieve, and manipulate patterns in text.
- Another great feature added in 10*g*R1 is the ability to receive warnings when code is compiled. I don't mean errors—we get these already. We can get warnings now using the `plsql_warnings` parameter, or the DBMS_WARNING package. They give us hints about potential performance problems and minor problems that do not result in errors at compile time.
- New datatypes—BINARY_FLOAT and BINARY_DOUBLE—are native floating-point datatypes that are an alternative to using the `NUMBER` type.
- DBMS_LOB offers support of large LOBs—between 8 and 128 terabytes (depending on block size). See Chapter 16 for more information.
- String literal customization. If you get tired of having to put two single quotes inside of a string literal, you can use `q'!...!'`, with the string placed inside the exclamation points. This lets you use one single quote in your string rather than requiring two. Here's a quick example using an anonymous block:

```
SET SERVEROUTPUT ON
BEGIN
   DBMS_OUTPUT.PUT_LINE('Ron's');
END;
/
```

This returns the following error:

```
ORA-01756: quoted string not properly terminated
```

To fix it, we used to have to use two single quotes in place of the apostrophe, as in `'Ron''s'`. 10*g*R1 provides another alternative:

```
BEGIN
   DBMS_OUTPUT.PUT_LINE(q'!Ron's!');
END;
/
```

This completes successfully and displays `Ron's` as intended.

Language Fundamentals

In this section we look at some of the basic features of PL/SQL, such as the ability to execute code without storing it, storing code for later use, and the differences between various types of stored objects. We discuss them at a high level, just to introduce the concepts. They are discussed in much greater detail in Chapters 3, 4, 8, and 9.

Anonymous Blocks

Anonymous blocks of code are not stored, and not named. They are executed in-session and cannot be called from another session. To execute the same code again, you must save the anonymous block to an OS file and run it, type it in again, or include it in a program that executes the block when needed.

You will find throughout the examples that anonymous blocks are used extensively. Anonymous blocks are perfect for scripting, or activities that you do not wish to repeat frequently. The following example is an anonymous block:

```
SET SERVEROUTPUT ON
DECLARE
   v_Date TIMESTAMP;
BEGIN
   SELECT systimestamp - 1/24
   INTO v_Date
   FROM dual;
   DBMS_OUTPUT.PUT_LINE('One hour ago: '||v_Date);
END;
/
```

The block begins with `DECLARE` or `BEGIN` and is not stored anywhere once executed.

NOTE
A PL/SQL block is a complete section of PL/SQL code. A PL/SQL program is made up of one or more blocks that logically divide the work. Blocks can even be nested within other blocks. Chapter 3 includes a full discussion on block structure.

Procedures

Procedures are named and stored. They can return a *value* when executed, but they *do not* have to. The only thing that must be returned is the success or failure of the execution.

Stored procedures, or named procedures, are given a unique name at creation time. They are owned by the user that created them unless otherwise stated in the creation script.

You can execute procedures from the SQL*Plus prompt, from within a SQL script, or from another PL/SQL block of code.

Functions

Functions differ from procedures in that they *must* return a value. Their structure is very similar to procedures, with the mandatory RETURN clause being the biggest difference. Functions are named and can be called from the SQL*Plus prompt, from within a SQL script, or from another PL/SQL block of code. When executing a function, you must have the ability to handle the value returned, though.

Packages

Packages are logical groupings of procedures and functions. They have two parts: the specification and the body.

The *specification,* or *spec,* is public and shows the structure of the package. When a package is described in SQL*Plus, it is the spec that is shown. The spec is always created or compiled before the body. In fact, it is possible to create the spec without ever creating the body.

Object Types

Oracle's object types allow you to write object-oriented code using PL/SQL. Object types are similar in structure to packages, having both a specification and a body. They provide a level of abstraction to your underlying data structure.

Object types may include attributes and methods. *Attributes* are defining characteristics of your object. A book, for example, might have attributes of title, number of pages, etc.

Methods act upon the underlying data structures for the object. All interaction between the application and object data should be done using methods.

Some of the advantages of using object types include

- **Abstraction** The application developer is removed from the relational data structures and thinks in terms of real-world structures.
- **Consistency** If all application interaction is done through objects rather than directly against the data structures, data corruption becomes much less likely to be introduced.
- **Simplicity** Instead of taking a real-world model and converting it to code, the model stays in the real world. If I want to know something about a book object, I look to the *book* object.

Features introduced since Oracle 9*i*R1 include inheritance, dynamic method dispatch, and type evolution. They make object-oriented programming using PL/SQL much more robust.

PL/SQL Statement Processing

When you execute a PL/SQL block, the code is passed to the PL/SQL engine. The engine may be in the data server itself or in one of the tools (like Oracle Reports) that bundles the PL/SQL engine with it. Next, the code is parsed, and the SQL is passed to the SQL engine, or SQL Statement Executor. The procedural statements are passed to the Procedural Statement Executor for processing.

Interpreted

Interpreted is the default mode for Oracle. This means that stored procedures, functions, and packages are compiled and stored as PL/SQL and are interpreted by Oracle (written in C) at run time. In interpreted mode, PL/SQL compilation is quicker, but code execution may be slower than if native compilation was used.

Native Compilation

Native compilation, first introduced in Oracle 9*i*R1 and improved in 10*g*R1, converts PL/SQL to C at compile time. This makes execution up to 30 percent faster, since no interpretation is required at run time.

Getting the Most from This Book

This book is fully revised and includes beginning, intermediate, and advanced topics. Sample code is used throughout to demonstrate features, and all of it is available online for download. The web site includes chapter directories, where all code referenced in the book is stored. The code for each chapter is intended to run independent of other chapters—no cross-chapter dependencies. Schema creation scripts are included for ease of testing. You will need to modify them as appropriate for your environment and database access.

There are some topics that require more space than we could possibly allocate inside the book. Instead of reducing coverage to fit the book, we have created supplemental papers for download that are extensions to chapter topics. We hope you will find this added coverage useful.

Audience

This book is written for new and experienced PL/SQL application developers, as well as for DBAs who would like to take advantage of all PL/SQL has to offer. Advanced chapters (11–17) require that you understand Chapters 1–10. If you are an experienced PL/SQL programmer, you may still want to peruse the first few chapters. We include discussions on new features, and example code that may generate some new ideas for your applications.

Regardless of your level of experience, we are confident that you will find something you had not yet discovered in each chapter.

Objective

PL/SQL is a mature, robust language that continues to improve with every release. As complexity increases, keeping up with new features becomes a daunting task. We aim to help you

- Learn PL/SQL if you are new to the language.
- Develop good form and efficient code.
- Understand features only referenced in passing in other texts, or not covered at all.
- Discover how powerful this language is!

Scope

Every book includes limitations. Ours are as follows:

- There is only limited coverage of database administration topics. If you wish to learn more about database administration, I will refer you to http://otn.oracle.com, or one of the excellent database administration books by Oracle Press.
- Performance tuning coverage is limited to making your PL/SQL efficient. It does not cover database performance tuning.
- We can only provide you with information, advice, and examples. It is up to you to take advantage of them, and write good code.

Assumptions

The base release for this book is Oracle 8.1.7.4, the terminal release of the Oracle 8*i* database. Coverage includes 8.1.7, 9*i*R1, 9*i*R2, and 10*g*R1. To take full advantage of this book, and the new features in Oracle, we recommend you download and install Oracle 10*g*R1 from OTN (http://otn.oracle.com). You can download the data server for free as long as you register (registration is also free).

TIP
10gR1 is a single-disk install, so the download will be much quicker than for any version of 9i.

At a minimum, it is recommended that you have access to an Oracle instance, and that you have the necessary permissions to create a user and the required objects. It is important that you are aware of your version of PL/SQL, since it impacts feature availability.

Since Oracle 8, PL/SQL versions have coincided with the database versions. In texts that cover releases prior to Oracle 8, you will see versioning like PL/SQL 1.1, 2.X, etc. To find the version you are running, query the V$VERSION view.

```
SELECT banner
FROM v$version;
```

The select returns the following in my current environment:

```
BANNER
--------------------------------
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - Prod
PL/SQL Release 10.1.0.2.0 - Production
CORE    10.1.0.2.0      Production
TNS for 32-bit Windows: Version 10.1.0.2.0 - Production
NLSRTL Version 10.1.0.2.0 - Production
```

So I am using version 10.1.0.2.0 of the database, and PL/SQL version 10.1.0.2 as well.

Conventions

We use different fonts through the book to highlight and differentiate certain text. Code examples, and external references to database objects in the text, are in COURIER. References to variables in the text are also in COURIER. Items of particular interest in a code example are placed in **bold** letters. Take special note of the Note and Tip sections in the book.

Examples

User creation scripts are included in each chapter that grant permissions required by the examples in that chapter alone. Do not use a schema creation script from one chapter for the examples in another.

Most chapters use example objects related to a bookstore. The base tables for most examples are BOOKS and AUTHORS, as shown earlier in this chapter in Figure 1-1. There are slight differences in the schema design between chapters in order to demonstrate different features.

The BOOKS table structure is

```
DESC books
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------
 ISBN                                      NOT NULL CHAR(10)
 CATEGORY                                           VARCHAR2(20)
 TITLE                                              VARCHAR2(100)
 NUM_PAGES                                          NUMBER
 PRICE                                              NUMBER
 COPYRIGHT                                          NUMBER(4)
 AUTHOR1                                            NUMBER
 AUTHOR2                                            NUMBER
 AUTHOR3                                            NUMBER
```

The AUTHORS table structure is as follows:

```
DESC authors
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------
 ID                                        NOT NULL NUMBER
 FIRST_NAME                                         VARCHAR2(50)
 LAST_NAME                                          VARCHAR2(50)
```

The schema creation script in Chapter 16, called `CreateLOBUser.sql`, creates two tablespaces to use for the storage parameter on a `create table` statement. The tablespace names and datafile names and locations can be modified as needed for your environment.

Finally, take time to review the different methods used to create the schemas and examples. We tried to employ different techniques throughout the book in order to demonstrate there is more than one way to accomplish the same task. As you look through the examples, think about how you can employ some of the same techniques, naming conventions, and strategies in your application design. Ingenuity, communication, attitude, discipline, and knowledge will propel you in your career using Oracle PL/SQL and aid in whatever task you apply yourself to.

Summary

In this chapter we introduced programming concepts, described how PL/SQL fits in both object-oriented and procedural programming categories, and previewed some of PL/SQL's features that are covered in this book. We reviewed the basics of relational databases and SQL, and looked at how PL/SQL compares with SQL and Java. Finally, we discussed this book, and how you can get the most out of it.

We hope you find this book helpful and discover things you never thought possible with PL/SQL.