第一部分　PL/SQL介绍及开发环境

第1章　PL/SQL介绍

PL/SQL是一种高级数据库程序设计语言，该语言专门用于在各种环境下对 Oracle数据库进行访问。由于该语言集成于数据库服务器中，所以 PL/SQL代码可以对数据进行快速高效的处理。除此之外，可以在Oracle数据库的某些客户端工具中，使用 PL/SQL语言也是该语言的一个特点。本章的主要内容是讨论引入 PL/SQL语言的必要性和该语言的主要特点，以及了解 PL/SQL语言的重要性和数据库版本问题。还要介绍一些贯穿全书的更详细的高级概念，并在本章的最后就我们在本书案例中使用的数据库表的若干约定做一说明。

1.1　为什么要引入PL/SQL语言

Oracle数据库是一种关系型数据库。通常我们把用于访问这种关系型数据库的程序设计语言叫做结构化查询语言，即SQL语言。SQL是一种灵活高效的查询语言，其主要功能是对关系数据库中的数据进行操作和处理。例如，下面的 SQL语句可以从数据库中将学习营养专业的全部学生一次删除：

```
DELETE FROM students
  WHERE major = 'Nutrition';
```

（本章的最后一节将对本书中使用的包括 students表在内的各种数据库表进行说明。）

SQL是先进的第四代程序设计语言，使用这种语言只需对要完成的任务进行描述，而不必指定实现任务的具体方法。以上面例子中的 DELETE语句为例，我们并不知道 SQL语言是如何找到学习营养专业的学生的。虽然按一般语言的做法推测，数据库服务器要按某种顺序逐个访问数据库表中的所有学生记录以决定删除满足条件的学生记录。但实际上，我们无法知道这些删除操作的细节。

第三代程序设计语言如 C语言和COBOL语言等是面向过程的语言。用第三代语言（ 3GL）编制的程序是一步一步地实现程序功能的。例如，我们可以用下面的程序段来实现上述的删除操作：

```
LOOP over each student record
  IF this record has major = 'Nutrition' THEN
    DELETE this record;
  END IF;
END LOOP;
```

面向对象的程序设计语言如 C++或Java也属于第三代程序设计语言。虽然这类语言采用了面向对象的程序结构，但程序中算法的实现还是要用各种语句逐步指定。

各种语言都有其自身的优缺点。相对于第三代程序设计语言来说，SQL一类的第四代程序语言使用起来非常简单，语言中语句的种类也比较少。但这类语言将用户与实际的数据结构和算法隔离开来，对数据的具体处理完全由该类语言的运行时系统实现。而在某些情况下，第三代语言使用的过程结构在表达某些程序过程来说是非常有用的。这也就是引入PL/SQL语言的原因，即PL/SQL语言将第四代语言的强大功能和灵活性与第三代语言的过程结构的优势融为一体。

PL/SQL代表面向过程化的语言与SQL语言的结合。我们可以从该语言的名称中看出，PL/SQL是在SQL语言中扩充了面向过程语言中使用的程序结构，如：

- 变量和类型（即可以予定义也可以由用户定义）
- 控制语句（如IF-THEN-ELSE）和循环
- 过程和函数
- 对象类型和方法（PL/SQL8.0版本以上）

PL/SQL语言实现了将过程结构与Oracle SQL的无缝集成，从而为用户提供了一种功能强大的结构化程序设计语言。例如，我们假设要修改一个学生的专业。如果没有该学生的记录的话，我们就为该学生创建一个新的记录。用PL/SQL编制的程序代码可以实现我们的要求，如下所示：

节选自在线代码3gl_4gL.sql

```
DECLARE
  /* Declare variables that will be used in SQL statements */
  v_NewMajor VARCHAR2(10) := 'History';
  v_FirstName VARCHAR2(10) := 'Scott';
  v_LastName VARCHAR2(10) := 'Urman';
BEGIN
  /* Update the students table. */
  UPDATE students
    SET major = v_NewMajor
    WHERE first_name = v_FirstName
    AND last_name = v_LastName;
  /* Check to see if the record was found. If not, then we need
     to insert this record. */
IF SQL%NOTFOUND THEN
  INSERT INTO students (ID, first_name, last_name, major)
    VALUES(student_sequence.NEXTVAL, v_FirstName, v_LastName,
           v_NewMajor);
  END IF;
END;
```

上面的例子中有两个不同的SQL语句（UPDATE和INSERT），这两个语句是第四代程序结构，同时该段程序中还使用了第三代语言的结构（变量声明和IF条件语句）。

注意 为了运行上面的程序例子，先要创建程序中引用的数据库对象（即表students和序列student_sequence）。可以使用本书CD-ROM中提供的脚本文件relTable.sql来实现上述工作。有关创建上述对象的进一步信息，请参见1.3.3节的内容。

PL/SQL语言在将SQL语言的灵活性及功能与第三代语言的可配置能力相结合方面是独一无

二的。该语言集成了面向过程语言的过程结构和强大的数据库操作，为设计复杂的数据库应用提供了功能强大、健壮可靠的程序设计语言。

1.1.1 PL/SQL与网络传输

目前的大多数数据库应用一般都采用客户 /服务器或三层模式。在客户 /服务器模式下，驻留在客户机上的应用程序使用 SQL语句向数据库服务器发送服务请求。通常，发送这种请求将导致过多的网络传输，如图 1-1左半部分的框图所示，每个 SQL语句将引起一次网络传输。现在将该图的左面与右面相比较，我们可以发现使用 PL/SQL语言可以将几个 SQL语句合并为一个 PL/SQL块发送给服务器，从而减少网络通信流量并提高应用程序的执行速度。

即使在客户机与服务器都运行在同一台设备上时，使用 PL/SQL编制的应用程序的性能也会有提高。在这种情况下，虽然不需要进行网络传输，但将 SQL语句打包传送将减少对数据库的访问次数。

PL/SQL语言的打包功能也适用于三层结构应用。在这种模式下，客户机（通常运行在 HTML浏览器下）与应用服务器进行通信，而应用服务器再与数据库交互操作， PL/SQL有利于应用服务器与数据库的通信。有关 PL/SQL的这种应用环境，我们将在本书的第 2章中讨论。

1.1.2 PL/SQL标准

Orcale数据库支持ANSI标准的SQL语言，即ANSI X3.135-1992文档中“数据库语言 SQL”定义的SQL语言。该标准通常称为 SQL92标准，只定义了 SQL语言本身。该标准并没有定义 PL/SQL所提供语言的3GL扩充内容。SQL92指定了三个实现层次：初等级别、中等级别和最高级别。Oracle7的7.0版（包括所有高版本的Oracle8和Oracle8i）实现了由美国国家标准技术研究所认证的SQL92标准的初等级别。现在 Oracle公司正在与美国国家标准协会 ANSI共同努力以确保Oracle数据库和PL/SQL语言的最新版本将实现SQL92的最高级别。

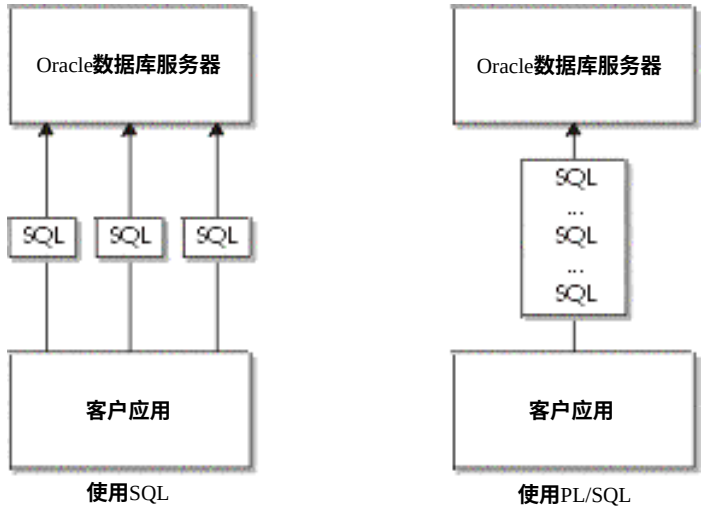


图1-1 在客户/服务器环境下使用SQL和PL/SQL的比较

1.2 PL/SQL的特点

介绍PL/SQL的不同特点与功能的最好方式是通过实际程序案例演示。本章下面几节将描述PL/SQL语言的若干主要特点。

1.2.1 PL/SQL的基本特点

本书的主要内容是介绍PL/SQL语言的高级功能。在下面几节中，我们将介绍该语言的基本特点。如果读者要进一步了解有关信息，请参考PL/SQL用户指南及索引手册，或参考《Oracle8 PL/SQL程序设计》一书。

1. PL/SQL的块结构

PL/SQL程序的基本结构是块。所有的PL/SQL程序都是由块组成的，这些块之间还可以相互嵌套。通常，程序中的每一块都实现一个逻辑操作，从而把不同的任务进行分割，由不同的块来实现。PL/SQL的块结构如下所示：

```
DECLARE
  /* Declarative section - PL/SQL variables, types, cursors,
     and local subprograms go here. */
BEGIN
  /* Executable section - procedural and SQL statements go here.
     This is the main section of the block and the only one
     that is required. */
EXCEPTION
  /* Exception-handling section - error-handling statements go
     here. */
END;
```

在上面演示的块结构中，只有执行部分是必须的，声明部分和异常处理部分都是可选的。块结构中的执行部分至少要有一个可执行语句。PL/SQL块采用的这种分段结构将PL/SQL程序的不同功能各自独立出来。

PL/SQL的这种特点是仿效第三代程序设计语言Ada采用的程序结构。在Ada语言中使用的很多程序结构包括Ada使用的块结构都适用于PL/SQL语言。除此之外，在PL/SQL语言中还可以发现Ada语言使用的异常处理方法、过程和函数声明以及包的定义的语法等特征。

2. 错误处理

PL/SQL块中的异常处理部分是用来响应应用程序运行中遇到的错误。把程序的主体部分与错误处理部分代码相互隔离，这样，程序的结构看起来十分清晰。例如，下面的PL/SQL块演示了将异常发生的时间及将遇到该异常错误的用户名记录在日志表的处理过程。

节选自在线代码Error.sql

```
DECLARE
  v_ErrorCode NUMBER; -- Code for the error
  v_ErrorMsg VARCHAR2(200); -- Message text for the error
  v_CurrentUser VARCHAR2(8); -- Current database user
  v_Information VARCHAR2(100); -- Information about the error
BEGIN
```

```
  /* Code that processes some data here */
EXCEPTION
WHEN OTHERS THEN
  -- Assign values to the log variables, using built-in
  -- functions.
  v_ErrorCode := SQLCODE;
  v_ErrorMsg := SQLERRM;
  v_CurrentUser := USER;
  v_Information := 'Error encountered on ' ||
    TO_CHAR(SYSDATE) || ' by database user ' || v_CurrentUser;
  -- Insert the log message into log_table.
  INSERT INTO log_table (code, message, info)
    VALUES (v_ErrorCode, v_ErrorMsg, v_Information);
END;
```

注意 上面的例子和许多其他例程都在本书的联机发布信息中。读者如要了解有关详细信息，请参阅1.3.3的内容。

3. 变量和类型

信息在数据库与PL/SQL程序之间是通过变量进行传递的。所谓变量就是可以由程序读取或赋值的存储单元。在上面的例子中， v_CurrentUser,v_ErrorCode,和v_Information都是变量。通常，变量是在PL/SQL块的声明部分定义的。

每个变量都有一个特定的类型与其关联。变量的类型定义了变量可以存放的信息类别。如下所示，PL/SQL变量可以与数据库列具有同样的类型：

```
DECLARE
  v_StudentName VARCHAR2(20);
  v_CurrentDate DATE;
  v_NumberCredits NUMBER(3);
```

PL/SQL变量也可以是其他类型：

```
DECLARE
  v_LoopCounter BINARY_INTEGER;
  v_CurrentlyRegistered BOOLEAN;
```

除此之外，PL/SQL还支持用户自定义的数据类型，如记录类型、表类型等。使用用户自定义的数据类型可以让你定制程序中使用的数据类型结构。下面是一个用户自定义的数据类型例子：

```
DECLARE
  TYPE t_StudentRecord IS RECORD (
    FirstName VARCHAR2(10),
    LastName VARCHAR2(10),
    CurrentCredits NUMBER(3)
  );
  v_Student t_StudentRecord;
```

4. 循环结构

PL/SQL支持多种循环结构。所谓循环就是指可以重复执行的同一代码段。例如，下面的程序块使用一个简单的循环来把数字1～50插入到表temp_table中：

节选自在线代码SimpleLoop.sql

```
DECLARE
  v_LoopCounter BINARY_INTEGER := 1;
BEGIN
  LOOP
    INSERT INTO temp_table (num_col)
      VALUES (v_LoopCounter);
    v_LoopCounter := v_LoopCounter + 1;
    EXIT WHEN v_LoopCounter > 50;
  END LOOP;
END;
```

PL/SQL中还提供了一种使用FOR的循环结构，这种循环的结构更简单。如下所示，我们可以使用这种FOR循环来实现上面的循环操作：

节选自在线代码NumricLoop.sql

```
BEGIN
  FOR v_LoopCounter IN 1..50 LOOP
    INSERT INTO temp_table (num_col)
      VALUES (v_LoopCounter);
  END LOOP;
END;
```

5. 游标

游标是用来处理使用SELECT语句从数据库中检索到的多行记录的工具。借助于游标的功能，数据库应用程序可以对一组记录逐个进行处理，每次处理一行。例如，下面的程序块可以检索到数据库中所有学生的名和姓：

节选自在线代码CursorLoop.sql

```
DECLARE
  v_FirstName VARCHAR2(20);
  v_LastName VARCHAR2(20);
  -- Cursor declaration. This defines the SQL statement to
  -- return the rows.
  CURSOR c_Students IS
  SELECT first_name, last_name
    FROM students;
BEGIN
  -- Begin cursor processing.
  OPEN c_Students;
  LOOP
    -- Retrieve one row.
    FETCH c_Students INTO v_FirstName, v_LastName;
    -- Exit the loop after all rows have been retrieved.
    EXIT WHEN c_Students%NOTFOUND;
    /* Process data here */
```

```
  END LOOP;
  -- End processing.
  CLOSE c_Students;
END;
```

1.2.2 PL/SQL的高级功能

下面将介绍几个有关PL/SQL高级功能的程序例子，这些高级功能将在本书的后几章做详细讲解。PL/SQL的高级功能是在其基本功能上实现的。

1. 过程和函数

PL/SQL中的过程和函数（通称为子程序）是PL/SQL块的一种特殊类型，这种类型的子程序可以以编译的形式存放在数据库中，并为后续的程序块调用。例如，下面的语句创建了一个叫做PrintStudents的过程，该过程使用DBMS_OUTPUT包将所有学生的姓名以定制的格式显示在屏幕上：

节选自在线代码PrintStudents.sql

```
CREATE OR REPLACE PROCEDURE PrintStudents(
  p_Major IN students.major%TYPE) AS

  CURSOR c_Students IS
    SELECT first_name, last_name
      FROM students
      WHERE major = p_Major;
BEGIN
  FOR v_StudentRec IN c_Students LOOP
    DBMS_OUTPUT.PUT_LINE(v_StudentRec.first_name || ' ' ||
                         v_StudentRec.last_name);
  END LOOP;
END;
```

一旦创建了该过程并将其存储在数据库中，我们就可以用如下所示的程序块来调用该过程：

节选自在线代码PrintStudents.sql

```
SQL> BEGIN
  2 PrintStudents('Computer Science');
  3 END;
  4 /
Scott Smith
Joanne Junebug
Shay Shariatpanahy
```

注意 使用SET SERVEROUTPUT ON命令可以设置DBMS_OUTPUT的输出功能。有关该命令的详细信息，请看见本书第2章的内容。

2. 包

PL/SQL中的子程序可以和变量与类型共同组成包。PL/SQL的包由两部分组成，即说明部分和包体。一个包可以带有多个相关的过程。例如，下面的包RoomsPkg就有两个过程，一个是插

入新教室信息的过程，另一个是从表rooms中删除一个教室的过程：

节选自在线代码RoomPkg.sql

```
CREATE OR REPLACE PACKAGE RoomsPkg AS
  PROCEDURE NewRoom(p_Building rooms.building%TYPE,
                    p_RoomNum rooms.room_number%TYPE,
                    p_NumSeats rooms.number_seats%TYPE,
                    p_Description rooms.description%TYPE);

  PROCEDURE DeleteRoom(p_RoomID IN rooms.room_id%TYPE);
END RoomsPkg;

CREATE OR REPLACE PACKAGE BODY RoomsPkg AS
  PROCEDURE NewRoom(p_Building rooms.building%TYPE,
                    p_RoomNum rooms.room_number%TYPE,
                    p_NumSeats rooms.number_seats%TYPE,
                    p_Description rooms.description%TYPE) IS
  BEGIN
    INSERT INTO rooms
      (room_id, building, room_number, number_seats, description)
      VALUES
      (room_sequence.NEXTVAL, p_Building, p_RoomNum, p_NumSeats,
      p_Description);
  END NewRoom;
  PROCEDURE DeleteRoom(p_RoomID IN rooms.room_id%TYPE) IS
  BEGIN
    DELETE FROM rooms
      WHERE room_id = p_RoomID;
  END DeleteRoom;
END RoomsPkg;
```

3. 动态SQL

借助于动态SQL，一个PL/SQL应用可以在运行期间构造并执行SQL语句。使用动态SQL方法有两种，一种是使用PL/SQL2.1版及以上支持的DBMS_SQL包，另一种是使用Oracle8i或更高版本支持的本地动态SQL。下面所示的DropTable就是使用DBMS_SQL实现的过程，其功能是释放指定的工作表：

节选自在线代码DropTable.sql

```
CREATE OR REPLACE PROCEDURE DropTable(p_Table IN VARCHAR2) AS
  v_SQLString VARCHAR2(100);
  v_Cursor BINARY_INTEGER;
  v_ReturnCode BINARY_INTEGER;
BEGIN
  -- Build the string based on the input parameter.
  v_SQLString := 'DROP TABLE ' || p_Table;

  -- Open the cursor.
```

```
  v_Cursor := DBMS_SQL.OPEN_CURSOR;

  -- Parse and execute the statement.
  DBMS_SQL.PARSE(v_Cursor, v_SQLString, DBMS_SQL.NATIVE);
  v_ReturnCode := DBMS_SQL.EXECUTE(v_Cursor);

  -- Close the cursor.
  DBMS_SQL.CLOSE_CURSOR(v_Cursor);
END DropTable;
```

如果使用Oracle8i或更高的版本，该过程可以使用本地动态SQL重写如下：

节选自在线代码PrintStudents.sql

```
CREATE OR REPLACE PROCEDURE DropTable(p_Table IN VARCHAR2) AS
  v_SQLString VARCHAR2(100);
BEGIN
  -- Build the string based on the input parameter.
  v_SQLString := 'DROP TABLE ' || p_Table;

  EXECUTE IMMEDIATE v_SQLString;
END DropTable;
```

4. **对象类型 （Oracle8及以上版本）**

Oracle8(包括PL/SQL 8)支持对象类型。Oracle中的对象类型由属性和方法组成并可以存储在数据库表中。下面的例子演示了创建对象类型的方法：

节选自在线代码12/objTypes.sql

```
CREATE OR REPLACE TYPE Student AS OBJECT (
  ID NUMBER(5),
  first_name VARCHAR2(20),
  last_name VARCHAR2(20),
  major VARCHAR2(30),
  current_credits NUMBER(3),

  -- Returns the first and last names, separated by a space.
  MEMBER FUNCTION FormattedName
    RETURN VARCHAR2,
  PRAGMA RESTRICT_REFERENCES(FormattedName, RNDS, WNDS, RNPS, WNPS),

  -- Updates the major to the specified value in p_NewMajor.
  MEMBER PROCEDURE ChangeMajor(p_NewMajor IN VARCHAR2),
  PRAGMA RESTRICT_REFERENCES(ChangeMajor, RNDS, WNDS, RNPS, WNPS),

  -- Updates the current_credits by adding the number of
  -- credits in p_CompletedClass to the current value.
  MEMBER PROCEDURE UpdateCredits(p_CompletedClass IN Class),
  PRAGMA RESTRICT_REFERENCES(UpdateCredits, RNDS, WNDS, RNPS, WNPS),
```

```
  -- ORDER function used to sort students.
  ORDER MEMBER FUNCTION CompareStudent(p_Student IN Student)
    RETURN NUMBER
);

CREATE OR REPLACE TYPE BODY Student AS
  MEMBER FUNCTION FormattedName
    RETURN VARCHAR2 IS
  BEGIN
    RETURN first_name || ' ' || last_name;
END FormattedName;

MEMBER PROCEDURE ChangeMajor(p_NewMajor IN VARCHAR2) IS
BEGIN
    major := p_NewMajor;
  END ChangeMajor;

MEMBER PROCEDURE UpdateCredits(p_CompletedClass IN Class) IS
BEGIN
    current_credits := current_credits +
    p_CompletedClass.num_credits;
  END UpdateCredits;
ORDER MEMBER FUNCTION CompareStudent(p_Student IN Student)
  RETURN NUMBER IS
BEGIN
  -- First compare by last names
  IF p_Student.last_name = SELF.last_name THEN
    -- If the last names are the same, then compare first names.
    IF p_Student.first_name < SELF.first_name THEN
      RETURN 1;
    ELSIF p_Student.first_name > SELF.first_name THEN
      RETURN -1;
    ELSE
      RETURN 0;
    END IF;
  ELSE
    IF p_Student.last_name < SELF.last_name THEN
      RETURN 1;
    ELSE
      RETURN -1;
   END IF;
  END IF;
 END CompareStudent;
END;
```

5. **集合**

PL/SQL**的集合类似于其他**3GL**中使用的数组。** PL/SQL**提供了三种不同的集合类型：按表索**

引（PL/SQL 2.0及更高版本）、嵌套表（PL/SQL 8.0及更高版本）、数组（PL/SQL 8.0及更高版本）。下面的程序例子介绍了上述三种集合类型的使用方法：

节选自在线代码Collections.sql

```
DECLARE
  TYPE t_IndexBy IS TABLE OF NUMBER
    INDEX BY BINARY_INTEGER;
  TYPE t_Nested IS TABLE OF NUMBER;
  TYPE t_Varray IS VARRAY(10) OF NUMBER;

  v_IndexBy t_IndexBy;
  v_Nested t_Nested;
  v_Varray t_Varray;
BEGIN
  v_IndexBy(1) := 1;
  v_IndexBy(2) := 2;
  v_Nested := t_Nested(1, 2, 3, 4, 5);
  v_Varray := t_Varray(1, 2);
END;
```

1.2.3 PL/SQL内置包

除了上述PL/SQL语言提供的功能外，Oracle也提供了若干具有特殊功能的内置包。本书自始至终将贯穿讨论这些功能的细节，并在本书的附录A中给予总结。下面列出了本书将重点讨论的内置包：

包	描　述
DBMS_ALERT	数据库报警，允许会话间通信
DBMS_JOB	任务调度服务
DBMS_LOB	大型对象操作
DBMS_PIPE	数据库管道，允许会话间通信
DBMS_SQL	动态SQL
UTL_FILE	文本文件的输入与输出

总的来说，所有DBMS-*包都存储在服务器中，只有包UTL_FILE既存储在服务器端又存储在客户端（某些客户环境，如Oracle表还提供了额外的包）。

1.3 本书的约定

本节介绍作者在本书中使用的几个约定。这些约定中包括用来标识PL/SQL版本的图标、引用Oracle文档资料的方法和在线案例所在的文件目录。

1.3.1 PL/SQL和Oracle 数据库版本说明

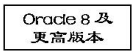
PL/SQL是随Oracle服务器一同提供的。PL/SQL的1.0版是与Oracle6.0版一起发行的。随后发

行的Oralce7提供了PL/SQL2.x系列版本。当Oracle8发布后，PL/SQL的版本号也升级到8系列。现在使用的Oracle8i（版本号为8.1）中的PL/SQL的版本号是8.1。将来发行的Oracle新版本数据库的PL/SQL版本号将与该数据库版本号保持一致。如表 1-1所示，该表列出了每版Oracle数据库与对应的PL/SQL语言新增的功能。本书的重点是讨论 PL/SQL2.0-8.1版的内容。本书使用以下的图标来表示某一版本具有的特殊功能：

该图标所在的段落将讨论PL/SQL 2.1版和更高版本具有的功能，如包 DBMS _SQL。

该图标所在的段落将讨论PL/SQL 2.2版和更高版本具有的功能，如游标变量。

该图标所在的段落将讨论 PL/SQL 2.3版和更高版本具有的功能，如包UTL _FILE。



该图标所在的段落将讨论PL/SQL 8.0版和更高版本具有的功能，如对象类型。

该图标所在的段落将讨论 PL/SQL 8.1版和更高版本具有的功能，如本地动态 SQL。

表1-1 Oracle和PL/SQL版本号与功能对照表

Oracle版本	PL/SQL版本	新增或变更的功能
6	1.0	第一版
7.0	2.0	数据类型CHAR变为定长
		存储子程序（包括过程、函数、包和触发器）
		用户定义的复合类型 —— 表和记录
		使用DBMS_PIPE和DBMS_ALERT包进行会话间通信
		在SQL *PLUS或服务器管理器中使用DBMS_OUTPUT包进行输出
7.1	2.1	用户定义的子类型
		在SQL语句中使用用户定义函数的功能
		使用DBMS_SQL包的动态PL/SQL
7.2	2.2	游标变量
		用户定义的约束子类型
		使用DBMS_JOB包调度PL/SQL批处理的功能
7.3	2.3	增强了游标变量的功能（扩充了对服务器的 fetch操作和弱类（weakly typed))
		使用UTL_FILE进行文件输入输出操作
		PL/SQL表属性和记录表
		以编译格式存储的触发器
8.0	8.0	对象类型和方法
		集合类型 —— 嵌套表和数组
		高级队列功能选项
		外部过程
		增强的LOB
8.1	8.1	本地动态SQL
		Java外部例程

（续）

Oracle版本	PL/SQL版本	新增或变更的功能
		调用权利 NOCOPY参数 自动事务 大容量处理

在使用PL/SQL语言编程时，确认PL/SQL的版本号将有助于使用该版本提供的相应先进功能。当与Oracle数据库连接时，初始化字符串中带有该数据库的版本号。请看下面两个连接显示字符串：

```
Connected to:
Oracle8 Enterprise Edition Release 8.0.6.0.0 - Production
With the Objects option
PL/SQL Release 8.0.6.0.0 - Production

and

Connected to:
Oracle8i Enterprise Edition Release 8.1.5.0.0 - Production
With the Partitioning and Java options
PL/SQL Release 8.1.5.0.0 - Production
```

这两个字符串都是合法的初始字符串。请注意在上面的显示中， PL/SQL的版本号是与数据库的版本号完全对应的。

本书的大部分案例程序是用运行在 SUN公司的Solaris 操作系统上的Oracle8.0.6实现的。本书的Oracle8i案例程序也是用运行在 SUN公司的Solaris 操作系统上的Oracle8i，8.1.5版编制的。本书所用的所有的计算机屏幕图形都是与服务器上的 Oracle数据库相连接的 Windows 95或Windows NT操作系统的窗口图形。

1.3.2 Oracle数据库文档

在本书的有关章节中，我向读者推荐了包含详细信息的 Oracle文档资料。由于这些文档手册的名称因数据库的版本不同而不同，我通常使用缩写的版本号来区别不同的手册。例如， Oracle服务器参考资料指的是 Oracle7、Oracle8或Oracle8i服务器参考资料，具体是指哪一本资料，这取决于你正在使用的Oracle数据库版本。

1.3.3 本书提供的CD-ROM内容简介

本书附加的CD-ROM中包括下列几类信息：

- 本书使用的程序案例的代码。读者也可以在 http://www.osborne.com站点浏览本书使用的这些程序案例。
- PL/SQL开发工具的五个试用版程序。有关这些开发工具的详细信息，参见本书的第2章及

第3章的内容。

- 本书第15章和第16章及附录A、附录B、附录C电子版内容。除此之外，本书第2章使用的屏幕图形也在CD-ROM中。

有关本书CD-ROM内容的详细信息，请读者阅读CD-ROM根目录下的超文本文件readme.html。

本书使用的程序案例在CD-ROM中的目录说明

本书中使用的程序案例的第一个注释行指出了该程序的名称。本书使用的所有程序案例的代码都存放在CD-ROM中CODE目录下以章节命名的子目录中。例如，请看下面我们在本章中使用的循环结构示范程序：

节选自在线代码SimpleLoop.sql

```
DECLARE
  v_LoopCounter BINARY_INTEGER := 1;
BEGIN
  LOOP
    INSERT INTO temp_table (num_col)
      VALUES (v_LoopCounter);
    v_LoopCounter := v_LoopCounter + 1;
    EXIT WHEN v_LoopCounter > 50;
  END LOOP;
END;
```

根据第一行的注释行信息，我们可以在CD-ROM中的目录code/ch01中找到该源程序文件simple.sql。CD-ROM中CODE目录中的readme.html文件对所有的程序案例都有说明。

1.4 本书案例使用的通用数据库表

本书中使用的程序案例是一个大学入学注册系统中的数据库表，这组表中最常用的表有三个：students、classes和rooms。这三个表中带有注册系统所需的记录。除了这三个主表之外，表registered_students带有已注册课程学生的信息。下面，我们来详细介绍这些表的结构。

注意 可以使用在线文档中的脚本relTable.sql来创建上述这些数据库表。Oracle8使用的对象类型表可由脚本objTables.sql来实现。上述两个脚本都存储在CODE子目录中。本节只介绍关系型表，有关对象表，请看objTables.sql的内容。

1. 序列

student_sequence序列用来生成表students主键的唯一健值，而room_sequence序列则是为表rooms的主健生成的唯一健值。

```
CREATE SEQUENCE student_sequence
  START WITH 10000
  INCREMENT BY 1;

CREATE SEQUENCE room_sequence
  START WITH 20000
```

```
  INCREMENT BY 1;
```

2. 表students

表students中包括了入学新生的有关信息。

```
CREATE TABLE students (
  id NUMBER(5) PRIMARY KEY,
  first_name VARCHAR2(20),
  last_name VARCHAR2(20),
  major VARCHAR2(30),
  current_credits NUMBER(3)
);

INSERT INTO students (id, first_name, last_name, major,
                      current_credits)
  VALUES (student_sequence.NEXTVAL, 'Scott', 'Smith',
          'Computer Science', 11);

INSERT INTO students (id, first_name, last_name, major,
                      current_credits)
  VALUES (student_sequence.NEXTVAL, 'Margaret', 'Mason',
          'History', 4);
INSERT INTO students (id, first_name, last_name, major,
                      current_credits)
  VALUES (student_sequence.NEXTVAL, 'Joanne', 'Junebug',
          'Computer Science', 8);

INSERT INTO students (id, first_name, last_name, major,
                      current_credits)
  VALUES (student_sequence.NEXTVAL, 'Manish', 'Murgratroid',
          'Economics', 8);

INSERT INTO students(id, first_name, last_name, major,
                      current_credits)
  VALUES(student_sequence.NEXTVAL, 'Patrick', 'Poll',
         'History', 4);

INSERT INTO students(id, first_name, last_name, major,
                      current_credits)
  VALUES (student_sequence.NEXTVAL, 'Timothy', 'Taller',
         'History', 4);

INSERT INTO students(id, first_name, last_name, major,
                      current_credits)
  VALUES (student_sequence.NEXTVAL, 'Barbara', 'Blues',
         'Economics', 7);

INSERT INTO students(id, first_name, last_name, major,
                      current_credits)
  VALUES (student_sequence.NEXTVAL, 'David', 'Dinsmore',
           'Music', 4);

INSERT INTO students(id, first_name, last_name, major,
```

```
                    current_credits)
  VALUES (student_sequence.NEXTVAL, 'Ester', 'Elegant',
         'Nutrition', 8);

INSERT INTO students(id, first_name, last_name, major,
                     current_credits)
  VALUES (student_sequence.NEXTVAL, 'Rose', 'Riznit',
          'Music', 7);

INSERT INTO STUDENTS(id, first_name, last_name, major,
                     current_credits)
  VALUES (student_sequence.NEXTVAL, 'Rita', 'Razmataz',
          'Nutrition', 8);

INSERT INTO students(id, first_name, last_name, major,
                    current_credits)

  VALUES (student_sequence.NEXTVAL, 'Shay', 'Shariatpanahy',
         'Computer Science', 3);
```

3. 表major_stats

该表中保存不同专业的统计信息。

```
CREATE TABLE major_stats (
  major VARCHAR2(30),
  total_credits NUMBER,
  total_students NUMBER);

INSERT INTO major_stats (major, total_credits, total_students)
  VALUES ('Computer Science', 22, 3);

INSERT INTO major_stats (major, total_credits, total_students)
  VALUES ('History', 12, 3);

INSERT INTO major_stats (major, total_credits, total_students)
  VALUES ('Economics', 15, 2);

INSERT INTO major_stats (major, total_credits, total_students)
  VALUES ('Music', 11, 2);

INSERT INTO major_stats (major, total_credits, total_students)
  VALUES ('Nutrition', 16, 2);
```

4. 表rooms

该表存储可用的教室有关信息。

```
CREATE TABLE rooms (
  room_id NUMBER(5) PRIMARY KEY,
  building VARCHAR2(15),
  room_number NUMBER(4),
  number_seats NUMBER(4),
  description VARCHAR2(50)
  );

INSERT INTO rooms (room_id, building, room_number, number_seats,
```

```
                    description)
  VALUES (room_sequence.NEXTVAL, 'Building 7', 201, 1000,
          'Large Lecture Hall');

INSERT INTO rooms (room_id, building, room_number, number_seats,
                    description)
  VALUES (room_sequence.NEXTVAL, 'Building 6', 101, 500,
           'Small Lecture Hall');

INSERT INTO rooms (room_id, building, room_number, number_seats,
                    description)
  VALUES (room_sequence.NEXTVAL, 'Building 6', 150, 50,
          'Discussion Room A');

INSERT INTO rooms (room_id, building, room_number, number_seats,
                    description)
  VALUES (room_sequence.NEXTVAL, 'Building 6', 160, 50,
          'Discussion Room B');

INSERT INTO rooms (room_id, building, room_number, number_seats,
                    description)
  VALUES (room_sequence.NEXTVAL, 'Building 6', 170, 50,
          'Discussion Room C');

INSERT INTO rooms (room_id, building, room_number, number_seats,
                    description)
  VALUES (room_sequence.NEXTVAL, 'Music Building', 100, 10,
          'Music Practice Room');

INSERT INTO rooms (room_id, building, room_number, number_seats,
                    description)
  VALUES (room_sequence.NEXTVAL, 'Music Building', 200, 1000,
          'Concert Room');

INSERT INTO rooms (room_id, building, room_number, number_seats,
                    description)
  VALUES (room_sequence.NEXTVAL, 'Building 7', 300, 75,
         'Discussion Room D');

INSERT INTO rooms (room_id, building, room_number, number_seats,
                    description)
  VALUES (room_sequence.NEXTVAL, 'Building 7', 310, 50,
          'Discussion Room E');
```

5. 表classes

该表描述了学生可以选择的课程。

```
CREATE TABLE classes (
  department CHAR(3),
  course NUMBER(3),
  description VARCHAR2(2000),
  max_students NUMBER(3),
  current_students NUMBER(3),
```

```
  num_credits NUMBER(1),
  room_id NUMBER(5),
  CONSTRAINT classes_department_course
    PRIMARY KEY (department, course),
CONSTRAINT classes_room_id
    FOREIGN KEY (room_id) REFERENCES rooms (room_id)
);
  INSERT INTO classes(department, course, description, max_students,
                      current_students, num_credits, room_id)
  VALUES ('HIS', 101, 'History 101', 30, 11, 4, 20000);

INSERT INTO classes(department, course, description, max_students,
                      current_students, num_credits, room_id)
  VALUES ('HIS', 301, 'History 301', 30, 0, 4, 20004);

INSERT INTO classes(department, course, description, max_students,
                      current_students, num_credits, room_id)
  VALUES ('CS', 101, 'Computer Science 101', 50, 0, 4, 20001);

INSERT INTO classes(department, course, description, max_students,
                      current_students, num_credits, room_id)
  VALUES ('ECN', 203, 'Economics 203', 15, 0, 3, 20002);

INSERT INTO classes(department, course, description, max_students,
                      current_students, num_credits, room_id)
  VALUES ('CS', 102, 'Computer Science 102', 35, 3, 4, 20003);

INSERT INTO classes(department, course, description, max_students,
                      current_students, num_credits, room_id)
  VALUES ('MUS', 410, 'Music 410', 5, 4, 3, 20005);

INSERT INTO classes(department, course, description, max_students,
                      current_students, num_credits, room_id)
  VALUES ('ECN', 101, 'Economics 101', 50, 0, 4, 20007);

INSERT INTO classes(department, course, description, max_students,
                      current_students, num_credits, room_id)
  VALUES ('NUT', 307, 'Nutrition 307', 20, 2, 4, 20008);

INSERT INTO classes(department, course, description, max_students,
                      current_students, num_credits, room_id)
  VALUES ('MUS', 100, 'Music 100', 100, 0, 3, NULL);
```

6. 表registered_students

该表保存学生目前参加的课程信息。

```
CREATE TABLE registered_students (
  student_id NUMBER(5) NOT NULL,
  department CHAR(3) NOT NULL,
  course NUMBER(3) NOT NULL,
  grade CHAR(1),
  CONSTRAINT rs_grade
    CHECK (grade IN ('A', 'B', 'C', 'D', 'E')),
```

```
  CONSTRAINT rs_student_id
    FOREIGN KEY (student_id) REFERENCES students (id),
  CONSTRAINT rs_department_course
    FOREIGN KEY (department, course)
    REFERENCES classes (department, course)
 );
INSERT INTO registered_students (student_id, department, course,
                                 grade)
  VALUES (10000, 'CS', 102, 'A');

INSERT INTO registered_students (student_id, department, course,
                                 grade)
  VALUES (10002, 'CS', 102, 'B');

INSERT INTO registered_students (student_id, department, course,
                                 grade)
  VALUES (10003, 'CS', 102, 'C');

INSERT INTO registered_students (student_id, department, course,
                                 grade)
  VALUES (10000, 'HIS', 101, 'A');

INSERT INTO registered_students (student_id, department, course,
                                 grade)
  VALUES (10001, 'HIS', 101, 'B');

INSERT INTO registered_students (student_id, department, course,
                                 grade)
  VALUES (10002, 'HIS', 101, 'B');

INSERT INTO registered_students (student_id, department, course,
                                 grade)
  VALUES (10003, 'HIS', 101, 'A');

INSERT INTO registered_students (student_id, department, course,
                                 grade)
  VALUES (10004, 'HIS', 101, 'C');

INSERT INTO registered_students (student_id, department, course,
                                 grade)
  VALUES (10005, 'HIS', 101, 'C');

INSERT INTO registered_students (student_id, department, course,
                                 grade)
  VALUES (10006, 'HIS', 101, 'E');

INSERT INTO registered_students (student_id, department, course,
                                 grade)
  VALUES (10007, 'HIS', 101, 'B');

INSERT INTO registered_students (student_id, department, course,
                                 grade)
  VALUES (10008, 'HIS', 101, 'A');
```

```
INSERT INTO registered_students (student_id, department, course,
                                 grade)
  VALUES (10009, 'HIS', 101, 'D');

INSERT INTO registered_students (student_id, department, course,
                                 grade)
  VALUES (10010, 'HIS', 101, 'A');

INSERT INTO registered_students (student_id, department, course,
                                 grade)
  VALUES (10008, 'NUT', 307, 'A');

INSERT INTO registered_students (student_id, department, course,
                                 grade)
  VALUES (10010, 'NUT', 307, 'A');

INSERT INTO registered_students (student_id, department, course,
                                 grade)
  VALUES (10009, 'MUS', 410, 'B');

INSERT INTO registered_students (student_id, department, course,
                                 grade)
  VALUES (10006, 'MUS', 410, 'E');

INSERT INTO registered_students (student_id, department, course,
                                 grade)
  VALUES (10011, 'MUS', 410, 'B');

INSERT INTO registered_students (student_id, department, course,
                                 grade)
  VALUES (10000, 'MUS', 410, 'B');
```

7. 表RS_audit

该表用来记录对表registered_students**所做的修改。**

```
CREATE TABLE RS_audit (
  change_type CHAR(1) NOT NULL,
  changed_by VARCHAR2(8) NOT NULL,
  timestamp DATE NOT NULL,
  old_student_id NUMBER(5),
  old_department CHAR(3),
  old_grade CHAR(1),
  new_student_id NUMBER(5),
  new_department CHAR(3),
  new_course NUMBER(3),
  new_grade CHAR(1)
  );
```

8. 表log_table

该表用来记录Oracle**数据库发生的错误信息。**

```
CREATE TABLE log_table (
  code NUMBER,
```

```
  message VARCHAR2(200),
  info    VARCHAR2(100)
  );
```

9. 表temp_table

该表用来存放临时数据。

```
CREATE TABLE temp_table (
  num_col  NUMBER,
  char_col VARCHAR2(60)
  );
```

10. 表connect_audit

该表由第6章的程序案例用来记录与数据库的连接和断开信息。

```
CREATE TABLE connect_audit (
  user_name VARCHAR2(30),
  operation VARCHAR2(30),
  timestamp DATE);
```

11. 表debug_table

该表由本书第3章的Debug包用来保存PL/SQL调试信息。

```
CREATE TABLE debug_table (
  linecount NUMBER,
  debug_str VARCHAR2(100)
  );
```

12. 表source和destination

这两个表是由第3章中调试程序案例使用的。

```
CREATE TABLE source (
  key NUMBER(5),
  value VARCHAR2(50) );

CREATE TABLE destination (
  key NUMBER(5),
  value NUMBER);
```

1.5 小结

我们在本章概述了引入PL/SQL语言的目的及该语言的主要特点。除此之外，我们还讨论了PL/SQL和数据库版本号的对应规则，介绍了本书CD-ROM中的内容和程序案例使用的数据库表。在下面的两章中，我们将讨论PL/SQL的各种开发、调试以及运行环境。