

Chapter
2

FUNDAMENTALS

*Get on the
Fast Track!*



TM

**SYS-ED/
COMPUTER
EDUCATION
TECHNIQUES, INC.**

Objectives

You will learn:

- C Rules and syntax of PL/SQL.
- C Block structure of the code.
- C How to define constants and variables and understand the scope of variables.
- C Creation of cursors and looping through the recordset.
- C Techniques for enhancing and simplifying declarations.
- C Coding techniques for interactive control.
- C To appreciate the benefits of using packages.
- C Coding and designing programs using subprograms.
- C Declaration and use of collections.
- C Handling and processing errors.

1 What is PL/SQL

With PL/SQL, SQL statements can be used for manipulating Oracle data and flow-of-control statements to process the data.

PL/SQL combines the data manipulating power of SQL with the data processing power of procedural languages.

With PL/SQL it is possible to:

- C declare constants and variables.
- C define procedures and functions.
- C trap runtime errors.

Example:

```
1  DECLARE
2  qty_on_hand  NUMBER(5);
3  BEGIN
4  SELECT quantity INTO qty_on_hand FROM inventory
5  WHERE product = 'TENNIS RACKET'
6  FOR UPDATE OF quantity;
7  IF qty_on_hand > 0 THEN -- check quantity
8  UPDATE inventory SET quantity = quantity - 1
9  WHERE product = 'TENNIS RACKET';
10 INSERT INTO purchase_record
11 VALUES ('Tennis racket purchased', SYSDATE);
12 ELSE
13 INSERT INTO purchase_record
14 VALUES ('Out of tennis rackets', SYSDATE);
15 END IF;
16 COMMIT;
17 END;
```

2 Block-Structured Language

PL/SQL is a block-structured language.

Procedures, functions, and anonymous blocks are the basic units that make up a PL/SQL program. They comprise a logical blocks, which can contain any number of nested sub-blocks.

A block or sub-block allows logically related declarations and statements to be grouped together..

Declarations are placed close to where they should be used. The declarations are local to the block and cease to exist when the block completes.

A PL/SQL block has three parts:

C	declarative	C	executable	C	exception-handling
---	-------------	---	------------	---	--------------------

Only the executable part is required.

First comes the declarative part, in which items can be declared. Once declared, items can be manipulated in the executable part. Exceptions raised during execution can be dealt with in the exception-handling part.

2.1 Block Structure

Sub-blocks can be nested in the executable and exception-handling parts of a PL/SQL block or subprogram, but not in the declarative part.

Local subprograms can be defined in the declarative part of any block.

Local subprograms can be called only from the block in which they are defined.

2.2 Variables and Constants

PL/SQL allows constants and variables to be declared; they then can be used in SQL and procedural statements anywhere an expression can be used. However, forward references are not allowed.

A constant or variable must be declared before referencing it in other statements, including other declarative statements.

Variables can have any SQL datatype, such as CHAR, DATE, or NUMBER, or any PL/SQL datatype, such as BOOLEAN or BINARY_INTEGER.

It is also possible to declare nested tables, variable-size arrays (varrays for short), and records using the TABLE, VARRAY, and RECORD composite datatypes.

3 Assigning Values to a Variable

Values can be assigned to a variable in two ways.

1. The assignment operator (:=), a colon followed by an equal sign.

The variable to the left of the operator and an expression to the right.

Example:

```
tax := price * tax_rate;
amount := TO_NUMBER(SUBSTR('750 dollars', 1, 3));
valid := FALSE;
```

2. Select or fetch database values into it.

Example:

```
SELECT sal * 0.10 INTO bonus FROM emp WHERE empno = emp_id;
```

3.1 Declaring Constants

Declaring a constant is similar to declaring a variable.

The major difference is that the keyword **CONSTANT** must be added and then immediately assigned a value to the constant.

No more assignments to the constant are allowed.

Example:

```
pi CONSTANT REAL := 3.14;
```

4 Cursors

Oracle uses work areas to execute SQL statements and store processing information.

PL/SQL's cursor construct allows a work area to be named and its stored information to be accessed.

There are two kinds of cursors:

C	implicit	C	explicit.
---	----------	---	-----------

PL/SQL implicitly declares a cursor for all SQL data manipulation statements, including queries that return only one row.

For queries that return more than one row, a cursor can be explicitly declared in order to process the rows individually.

Example:

```
1 DECLARE
2     CURSOR c1 IS
3         SELECT empno, ename, job FROM emp WHERE deptno = 20;
```

A PL/SQL program opens a cursor, processes rows returned by a query, then closes the cursor.

A cursor marks the current position in a result set.

The OPEN, FETCH, and CLOSE statements are used for controlling a cursor.

- C The OPEN statement executes the query associated with the cursor, identifies the result set, and positions the cursor before the first row.
- C The FETCH statement retrieves the current row and advances the cursor to the next row.
- C When the last row has been processed, the CLOSE statement disables the cursor.

5 Cursor FOR Loops

In most situations that require an explicit cursor, you can simplify coding by using a cursor FOR loop instead of the OPEN, FETCH, and CLOSE statements.

A cursor FOR loop implicitly declares its loop index as a record that represents a row fetched from the database.

Next, it opens a cursor, repeatedly fetches rows of values from the result set into fields in the record, then closes the cursor when all rows have been processed.

Example:

```
1 DECLARE
2     CURSOR c1 IS
3         SELECT ename, sal, hiredate, deptno FROM emp;
4     ...
5 BEGIN
6     FOR emp_rec IN c1 LOOP
7         ...
8         salary_total := salary_total + emp_rec.sal;
9     END LOOP;
```

6 Attributes

PL/SQL variables and cursors have attributes, which are properties which reference the datatype and structure of an item without repeating its definition.

Database columns and tables have similar attributes, which you can use to ease maintenance. A percent sign (%) serves as the attribute indicator.

6.1 %TYPE

The %TYPE attribute provides the datatype of a variable or database column. This is particularly useful when declaring variables that will hold database values.

To declare a variable named `my_title` that has the same datatype as column `title`, use dot notation and the %TYPE attribute.

Example:

```
my_title books.title%TYPE;
```

Declaring `my_title` with %TYPE has two advantages:

- C It is not necessary to know the exact datatype of `title`.
- C When changing the database definition of `title` such as making it a longer character string, the datatype of `my_title` changes accordingly at run time.

6.2 %ROWTYPE

In PL/SQL, records are used to group data. A record consists of a number of related fields in which data values can be stored. The %ROWTYPE attribute provides a record type that represents a row in a table.

The record can store an entire row of data selected from the table or fetched from a cursor or cursor variable.

Columns in a row and corresponding fields in a record have the same names and datatypes.

In the example below, you declare a record named `dept_rec`. Its fields have the same names and datatypes as the columns in the `dept` table.

```
1 DECLARE
2     dept_rec dept%ROWTYPE; -- declare record variable
```

Dot notation is used for referencing fields.

Example:

```
my_deptno := dept_rec.deptno;
```

If a cursor has been declared that retrieves the last name, salary, hire date, and job title of an employee, the %ROWTYPE can be used for declaring a record that stores the same information.

Example:

```
1 DECLARE
2     CURSOR c1 IS
3         SELECT ename, sal, hiredate, job FROM emp;
4     emp_rec c1%ROWTYPE; -- declare record variable that represents
5                          -- a row fetched from the emp table
```

When executing the statement:

```
FETCH c1 INTO emp_rec;
```

the value in the ename column of the emp table is assigned to the ename field of emp_rec, the value in the sal column is assigned to the sal field, and so on.

7 Control Structures

PL/SQL can be used for manipulating Oracle data.

PL/SQL processes the data using conditional, iterative, and sequential flow-of-control statements such as IF-THEN-ELSE, FOR-LOOP, WHILE-LOOP, EXIT-WHEN, and GOTO.

7.1 Conditional Control

The IF-THEN-ELSE statement provides for the execution of a sequence of statements conditionally.

- C The IF clause checks a condition.
- C the THEN clause defines what to do if the condition is true.
- C the ELSE clause defines what to do if the condition is false or null.

A sequence of statements that uses query results to select alternative actions is common in database applications.

Another common sequence inserts or deletes a row only if an associated entry is found in another table.

These common sequences can be bundled into a PL/SQL block using conditional logic.

7.2 Iterative Control

LOOP statements provides for the execution of a sequence of statements multiple times.

The LOOP keyword is placed before the first statement in the sequence and the keywords END LOOP after the last statement in the sequence.

Example:

```
1     LOOP
2     -- sequence of statements
3     END LOOP;
```

This code repeats a sequence of statements continually.

The FOR-LOOP statement provides for a range of integers to be specified, then execute a sequence of statements once for each integer in the range.

Example:

```
1     FOR i IN 1..order_qty LOOP
2         UPDATE sales SET custno = customer_id
3             WHERE serial_num = serial_num_seq.NEXTVAL;
4     END LOOP;
```

The WHILE-LOOP statement associates a condition with a sequence of statements. Before each iteration of the loop, the condition is evaluated.

- C If the condition is true, the sequence of statements is executed, then control resumes at the top of the loop.
- C If the condition is false or null, the loop is bypassed and control passes to the next statement.

Example:

```
1     -- available online in file 'examp3'
2     DECLARE
3         salary          emp.sal%TYPE;
4         mgr_num         emp.mgr%TYPE;
5         last_name       emp.ename%TYPE;
6         starting_empno CONSTANT NUMBER(4) := 7902;
7     BEGIN
8         SELECT sal, mgr INTO salary, mgr_num FROM emp
9             WHERE empno = starting_empno;
10        WHILE salary < 4000 LOOP
11            SELECT sal, mgr, ename INTO salary, mgr_num, last_name
12                FROM emp WHERE empno = mgr_num;
13        END LOOP;
14        INSERT INTO temp VALUES (NULL, salary, last_name);
15        COMMIT;
16    END;
```

The EXIT-WHEN statement allows for the completion of a loop even if further processing is impossible or undesirable.

When the EXIT statement is encountered, the condition in the WHEN clause is evaluated. If the condition is true, the loop completes and control passes to the next statement.

Example:

```
1 LOOP
2   ...
3   total := total + salary;
4   EXIT WHEN total > 25000; -- exit loop if condition is true
5 END LOOP;
6 -- control resumes here
7 Sequential Control
```

The GOTO statement provides for the branching to a label unconditionally.

The label, an undeclared identifier enclosed by double angle brackets, must precede an executable statement or a PL/SQL block.

When executed, the GOTO statement transfers control to the labeled statement or block.

Example:

```
1 IF rating > 90 THEN
2   GOTO calc_raise; -- branch to label
3 END IF;
4 ...
5 <<calc_raise>>
6 IF job_title = 'SALESMAN' THEN -- control resumes here
7   amount := commission * 0.25;
8 ELSE
9   amount := salary * 0.10;
10 END IF;
```

8 Subprograms

PL/SQL has two types of subprograms called procedures and functions, which can take parameters and be invoked.

A subprogram is similar to a miniature program, beginning with a header followed by an optional declarative part, an executable part, and an optional exception-handling part.

Example:

```
1      PROCEDURE award_bonus (emp_id NUMBER) IS
2          bonus          REAL;
3          comm_missing EXCEPTION;
4      BEGIN
5          SELECT comm * 0.15 INTO bonus FROM emp WHERE empno = emp_id;
6          IF bonus IS NULL THEN
7              RAISE comm_missing;
8          ELSE
9              UPDATE payroll SET pay = pay + bonus WHERE empno = emp_id;
10         END IF;
11     EXCEPTION
12         WHEN comm_missing THEN
13             ...
14     END award_bonus;
```

9 Packages

PL/SQL provides for the bundling of logically related types, variables, cursors, and subprograms into a package.

Each package is easy to understand and the interfaces between packages are simple, clear, and well defined. This aids application development.

Packages usually have two parts:

C	a specification	C	a body
---	-----------------	---	--------

The specification is the interface to applications; it declares the types, constants, variables, exceptions, cursors, and subprograms available for use. The body defines cursors and subprograms and so implements the specification.

Example:

```
1 CREATE PACKAGE emp_actions AS -- package specification
2   PROCEDURE hire_employee (empno NUMBER, ename CHAR, ...);
3   PROCEDURE fire_employee (emp_id NUMBER);
4 END emp_actions;
5
6 CREATE PACKAGE BODY emp_actions AS -- package body
7   PROCEDURE hire_employee (empno NUMBER, ename CHAR, ...) IS
8   BEGIN
9     INSERT INTO emp VALUES (empno, ename, ...);
10  END hire_employee;
11  PROCEDURE fire_employee (emp_id NUMBER) IS
12  BEGIN
13    DELETE FROM emp WHERE empno = emp_id;
14  END fire_employee;
15 END emp_actions;
```

In this code two employment procedures have been packaged.

Only the declarations in the package specification are visible and accessible to applications. Implementation details in the package body are hidden and inaccessible.

Packages can be compiled and stored in an Oracle database, where their contents can be shared by many applications.

When you call a packaged subprogram for the first time, the whole package is loaded into memory. So, subsequent calls to related subprograms in the package require no disk I/O.

10 Objects and Collection

10.1 Data Abstraction

Data abstraction provides for the extraction of essential properties of data while ignoring unnecessary details.

Once a data structure has been designed, it will be easier to focus on designing algorithms that manipulate the data structure.

10.2 Collections

The collection types `TABLE` and `VARRAY` allow nested tables and variable-size arrays (varrays for short) to be declared.

A collection is an ordered group of elements, all of the same type. Each element has a unique subscript that determines its position in the collection.

To reference an element, use standard subscripting syntax.

Collections work like the arrays found in most third-generation programming languages.

Collections can be passed as parameters.

10.3 Records

The `%ROWTYPE` attribute is used for declaring a record that represents a row in a table or a row fetched from a cursor. But, with a user-defined record, it is possible to declare fields.

Records contain uniquely named fields, which can have different datatypes.

10.4 Object Types

In PL/SQL, object-oriented programming is based on object types. An object type encapsulates a data structure along with the functions and procedures needed to manipulate the data.

The variables that form the data structure are called attributes. The functions and procedures that characterize the behavior of the object type are called methods.

Object types reduce complexity by breaking down a large system into logical entities. This allows software components to be created which are modular, maintainable, and reusable.

11 Error Handling

PL/SQL makes it easy to detect and process predefined and user-defined error conditions called exceptions.

When an error occurs, an exception is raised. That is, normal execution stops and control transfers to the exception-handling part of your PL/SQL block or subprogram.

To handle raised exceptions, separate routines called exception handlers can be written.

Predefined exceptions are raised implicitly by the runtime system.

User-defined exceptions must be raised explicitly with the RAISE statement.