

Database Management System Lab (CS 29006)

KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY

School of Computer Engineering



Strictly for internal circulation (within KIIT) and reference only. Not for outside circulation without permission

1 Credit

Course Committee

Lab Contents



2

Sr #	Major and Detailed Coverage Area	Lab#
1	<p>PL/SQL Programming Language</p> <ul style="list-style-type: none"><input type="checkbox"/> Notable Facts<input type="checkbox"/> Features<input type="checkbox"/> Basic Syntax<input type="checkbox"/> Conditions<input type="checkbox"/> Loops	8

PL/SQL



3

The PL/SQL programming language was developed by Oracle Corporation in the late 1980s as procedural extension language for SQL and the Oracle relational database.

Following are **notable facts** about PL/SQL:

- ❑ PL/SQL is a completely portable, high-performance transaction-processing language.
- ❑ PL/SQL provides a built-in interpreted and OS independent programming environment.
- ❑ PL/SQL can also directly be called from the command-line SQL*Plus interface.
- ❑ PL/SQL's general syntax is based on that of ADA and Pascal programming language.
- ❑ Apart from Oracle, PL/SQL is available in TimesTen in-memory database and IBM DB2.

PL/SQL Feature



4

PL/SQL has the following features:

- ☐ PL/SQL is tightly integrated with SQL.
- ☐ It offers extensive error checking.
- ☐ It offers numerous data types.
- ☐ It offers a variety of programming structures.
- ☐ It supports structured programming through functions and procedures.
- ☐ It supports object-oriented programming.
- ☐ It supports developing web applications and server pages.

PL/SQL Basic Syntax



5

PL/SQL is a block-structured language, meaning that PL/SQL programs are divided and written in logical blocks of code. Each block consists of three sub-parts:

Sr #	Sections & Description
1	Declarations This section starts with the keyword DECLARE. It is an optional section and defines all variables, cursors, subprograms, and other elements to be used in the program.
2	Executable Commands This section is enclosed between the keywords BEGIN and END and it is a mandatory section. It consists of the executable PL/SQL statements of the program. It should have at least one executable line of code, which may be just a NULL command to indicate that nothing should be executed.
3	Exception Handling This section starts with the keyword EXCEPTION. This section is again optional and contains exception(s) that handle errors in the program.

PL/SQL Syntax cont...



6

Every PL/SQL statement ends with a semicolon (;). PL/SQL blocks can be nested within other PL/SQL blocks using BEGIN and END.

Basic Structure

DECLARE

<declarations section>

BEGIN

<executable command(s)>

EXCEPTION

<exception handling>

END;

Note

The **END;** line signals the end of the PL/SQL block. To run the code from SQL command line, you may need to type / at the beginning of the first blank line after the last line of the code. If the output message is not displaying, **set serveroutput on** command to be executed to hint Oracle to dump/flush server output to the client's buffer and client will read from this buffer to display output accordingly.

Hello World Example

DECLARE

message varchar2(20):= 'Hello, World!';

BEGIN

dbms_output.put_line(message);

END;

/

PL/SQL Examples



7

Example 1

```
DECLARE
    salutation char(20);
    greetings varchar2(20);
BEGIN
    salutation := 'Reader ';
    greetings := 'Welcome to the World';
    dbms_output.put_line('Hello ' || salutation || greetings);
END;
/
```

Example 2

```
DECLARE
    a integer := 10;
    b integer := 20;
    c integer;
    f real;
BEGIN
    c := a + b;
    dbms_output.put_line('Value of c: ' || c);
    f := 70.0/3.0;
    dbms_output.put_line('Value of f: ' || f);
END;
/
```

PL/SQL Example 4



8

```
DECLARE
```

```
-- Global variables
```

```
num1 number := 95;
```

```
num2 number := 85;
```

```
BEGIN
```

```
dbms_output.put_line('Outer Variable num1: ' || num1);
```

```
dbms_output.put_line('Outer Variable num2: ' || num2);
```

```
DECLARE
```

```
-- Local variables
```

```
num1 number := 195;
```

```
num2 number := 185;
```

```
BEGIN
```

```
dbms_output.put_line('Inner Variable num1: ' || num1);
```

```
dbms_output.put_line('Inner Variable num2: ' || num2);
```

```
END;
```

```
END;
```

```
/
```


PL/SQL Example 5



9

```
DECLARE
```

```
-- Global variables
```

```
num1 number;
```

```
num2 number;
```

```
BEGIN
```

```
/* Taking value from user */
```

```
num1 := &num1;
```

```
num2 := &num2;
```

```
/* printing the user supplied value */
```

```
dbms_output.put_line('Variable num1: ' || num1);
```

```
dbms_output.put_line('Variable num2: ' || num2);
```

```
END;
```

```
/
```

Assigning SQL Query Results to PL/SQL Variables



10

SELECT INTO statement of SQL can be used to assign values to PL/SQL variables. For each item in the SELECT list, there must be a corresponding, type-compatible variable in the INTO list. The following example illustrates the concept.

Table Creation

```
CREATE TABLE CUSTOMER
(
  ID   INT NOT NULL,
  NAME VARCHAR(20) NOT NULL,
  AGE  INT NOT NULL,
  ADDRESS CHAR(25),
  SALARY  DECIMAL(18,2),
  PRIMARY KEY (ID)
);
```

Data Insertion

```
INSERT INTO CUSTOMER (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (1, 'Ramesh', 32, 'Ahmedabad', 2000.00 );
INSERT INTO CUSTOMER (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (2, 'Khilan', 25, 'Delhi', 1500.00 );
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (3, 'kaushik', 23, 'Kota', 2000.00 );
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (4, 'Chaitali', 25, 'Mumbai', 6500.00 );
```

Assigning values from the above table to PL/SQL variables

```
DECLARE
  c_id customer.id%type := 1;
  c_name customer.name%type;
  c_addr customer.address%type;
```

```
BEGIN
  SELECT name, address INTO c_name,
  c_addr FROM customer  WHERE id = c_id;
  dbms_output.put_line('Customer ' || c_name || ' from ' || c_addr);
```

```
END;
/
```

Const Declaration



11

A constant is declared using the `CONSTANT` keyword. It requires an initial value and does not allow that value to be changed. For example:

`DECLARE`

`-- constant declaration`

`pi constant number := 3.141592654;`

`-- other declarations`

`radius number(5,2);`

`dia number(5,2);`

`circumference number(7, 2);`

`area number (10, 2);`

`BEGIN`

`-- processing`

`radius := &radius;`

`dia := radius * 2;`

`circumference := 2.0 * pi * radius;`

`area := pi * radius * radius;`

`-- output`

`dbms_output.put_line('Radius: ' || radius);`

`dbms_output.put_line('Diameter: ' || dia);`

`dbms_output.put_line('Circumference: ' || circumference);`

`dbms_output.put_line('Area: ' || area);`

`END;`

`/`

PL/SQL Conditions



12

Decision-making structures require that the programmer specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false. Following is the general form of a typical conditional (i.e., decision making) structure found in most of the programming languages:

PL/SQL programming language provides following types of decision-making statements.

1. IF - THEN statement
2. IF-THEN-ELSE statement
3. IF-THEN-ELSIF statement
4. Case statement
5. Nested IF-THEN-ELSE

IF - THEN statement



13

Syntax:

IF condition THEN

Statement 1;

Statement 2;

.....

END IF;

Example 1:

DECLARE

a number(2);

BEGIN

a:= &a;

IF(a < 20) THEN

dbms_output.put_line('a is less than 20 ');

END IF;

dbms_output.put_line('value of a is : ' || a);

END;

/

Example 2:

DECLARE

c_id customer.id%type := 1;

c_sal customer.salary%type;

BEGIN

SELECT salary INTO c_sal

FROM customer WHERE id = c_id;

IF (c_sal <= 2000) THEN

UPDATE customer

SET salary = salary + 1000

WHERE id = c_id;

dbms_output.put_line ('Salary updated');

END IF;

END;

/

IF-THEN-ELSE statement



14

Syntax

IF condition THEN

Statement 1;

Statement 2;

...

ELSE

Statement 3;

Statement 4;

...

END IF;

Example

DECLARE

a number(3) := 100;

BEGIN

IF(a < 20) THEN

dbms_output.put_line('a is less than 20 ');

ELSE

dbms_output.put_line('a is not less than 20 ');

END IF;

dbms_output.put_line('value of a is : ' || a);

END;

/

Class Work

Write A PL/SQL Program to check whether a user supplied year is leap year or not.

IF-THEN-ELSIF statement



15

Syntax

```
IF(boolean_expression 1) THEN  
    Statement 1;  
ELSIF( boolean_expression 2) THEN  
    Statement 2;  
ELSIF( boolean_expression 3) THEN  
    Statement 3;  
ELSE  
    Statement 4;  
END IF;
```

Example

```
DECLARE  
    a number(3) := 100;  
BEGIN  
    IF ( a = 10 ) THEN  
        dbms_output.put_line('Value of a is 10' );  
    ELSIF ( a = 20 ) THEN  
        dbms_output.put_line('Value of a is 20' );  
    ELSIF ( a = 30 ) THEN  
        dbms_output.put_line('Value of a is 30' );  
    ELSE  
        dbms_output.put_line('None of the values is matching');  
    END IF;  
    dbms_output.put_line('Exact value of a is: '|| a );  
END;  
/
```

CASE statement



16

Syntax

CASE selector

WHEN 'value1' THEN Stat 1;

WHEN 'value2' THEN Stat 2;

WHEN 'value3' THEN Stat 3;

...

ELSE Stat n; -- default case

END CASE;

Example

DECLARE

grade char(1) := 'A';

BEGIN

CASE grade

when 'O' then dbms_output.put_line('Outstanding');

when 'E' then dbms_output.put_line('Excellent');

when 'A' then dbms_output.put_line('Very good');

when 'B' then dbms_output.put_line('Good');

when 'C' then dbms_output.put_line('Fair');

when 'D' then dbms_output.put_line('Pass');

when 'F' then dbms_output.put_line('Better try again');

else dbms_output.put_line('No such grade');

END CASE;

END;

/

Nested IF-THEN-ELSE statement



17

Syntax

IF(boolean_expression 1)THEN

-- executes when the boolean expression 1 is true

IF(boolean_expression 2) THEN

-- executes when the boolean expression 2 is true

sequence-of-statements;

END IF;

ELSE

-- executes when the boolean expression 1 is not true

else-statements;

END IF;

Example

DECLARE

a number(3) := 100;

b number(3) := 200;

BEGIN

IF(a = 100) THEN

-- if condition is true then check the following

IF(b = 200) THEN

-- if condition is true then print the following

dbms_output.put_line('Value of a is 100 and b is 200');

END IF;

END IF;

dbms_output.put_line('Exact value of a is : ' || a);

dbms_output.put_line('Exact value of b is : ' || b);

END;

/

Loops



18

There may be a situation when you need to execute a block of code several number of times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on. Programming languages provide various control structures that allow for more complicated execution paths. A loop statement allows to execute a statement or group of statements multiple times.

PL/SQL programming language provides following types of looping statements.

1. Basic Loop
2. While Loop
3. For Loop
4. Nested Loop

Basic Loop



19

Basic loop structure encloses sequence of statements in between the LOOP and END LOOP statements. With each iteration, the sequence of statements is executed and then control resumes at the top of the loop. An EXIT statement or an EXIT WHEN statement is required to break the loop.

Syntax

LOOP

Sequence of statements;

END LOOP;

Example with exit

```
DECLARE
  x number := 10;
BEGIN
  LOOP
    dbms_output.put_line(x);
    x := x + 10;
    IF x > 50 THEN
      exit;
    END IF;
  END LOOP;
  -- after exit, control resumes here
  dbms_output.put_line('After Exit x is: ' || x); /
END;
```

Example with exit when

```
DECLARE
  x number := 10;
BEGIN
  LOOP
    dbms_output.put_line(x);
    x := x + 10;
    exit WHEN x > 50;
  END LOOP;
  -- after exit, control resumes here
  dbms_output.put_line('After Exit x is: ' || x);
END;
```

While Loop



20

A WHILE loop statement in PL/SQL programming language repeatedly executes a target statement as long as a given condition is true.

Syntax

```
WHILE condition LOOP  
  sequence_of_statements  
END LOOP;
```

Example 1

```
DECLARE  
  a number(2) := 10;  
BEGIN  
  WHILE a < 20 LOOP  
    dbms_output.put_line('value of a: ' || a);  
    a := a + 1;  
  END LOOP;  
END;  
/
```

Example 2

```
DECLARE  
  a number(2) := 20;  
BEGIN  
  WHILE a < 10 LOOP  
    dbms_output.put_line('value of a: ' || a);  
    a := a - 1;  
  END LOOP;  
END;  
/
```

For and Reverse For Loop



21

A FOR loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

For Loop Syntax

FOR counter IN initial_value .. final_value LOOP

sequence_of_statements;

END LOOP;

Reverse For Loop

By default, iteration proceeds from the initial value to the final value, generally upward from the lower bound to the higher bound. You can reverse this order by using the REVERSE keyword. In such case, iteration proceeds the other way. After each iteration, the loop counter is decremented. However, you must write the range bounds in ascending (not descending) order.

For Loop Example

```
DECLARE
  a number(2);
BEGIN
  FOR a in 10 .. 20 LOOP
    dbms_output.put_line('value of a: ' || a);
  END LOOP;
END;
/
```

Reverse For Loop Example

```
DECLARE
  a number(2);
BEGIN
  FOR a IN REVERSE 10 .. 20 LOOP
    dbms_output.put_line('value of a: ' || a);
  END LOOP;
END;
/
```

Nested Loops



22

PL/SQL allows using one loop inside another loop. Following section shows few examples to illustrate the concept.

Syntax - nested basic loop

LOOP

Sequence of statements1

LOOP

Sequence of statements2

END LOOP;

END LOOP;

Syntax - nested while loop

WHILE condition1 LOOP

sequence_of_statements1

WHILE condition2 LOOP

sequence_of_statements2

END LOOP;

END LOOP;

Syntax - nested for loop

FOR counter1 IN initial_value1 .. final_value1 LOOP

sequence_of_statements1

FOR counter2 IN initial_value2 .. final_value2 LOOP

sequence_of_statements2

END LOOP;

END LOOP;

Example



Nested Loops

Labeling a PL/SQL Loop



23

PL/SQL loops can be labeled. The label should be enclosed by double angle brackets (<< and >>) and appear at the beginning of the LOOP statement. The label name can also appear at the end of the LOOP statement. You may use the label in the EXIT statement to exit from the loop.

Example

```
DECLARE
    i number(1);
    j number(1);
BEGIN
    << outer_loop >>
    FOR i IN 1..3 LOOP
        << inner_loop >>
        FOR j IN 1..3 LOOP
            dbms_output.put_line('i is: ' || i || ' and j is: ' || j);
        END loop inner_loop;
    END loop outer_loop;
END;
```

/

Loop Control Structure



24

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed. PL/SQL supports the following control statements.

- ☐ Exit ☐ Exit When
- ☐ Continue ☐ Goto

Exit

The EXIT statement in PL/SQL programming language has following two usages:

- ☐ When the EXIT statement is encountered inside a loop, the loop is immediately terminated and program control resumes at the next statement following the loop.
- ☐ If you are using nested loops (i.e. one loop inside another loop), the EXIT statement will stop the execution of the innermost loop and start executing the next line of code after the block.

Exit When

The EXIT-WHEN statement allows the condition in the WHEN clause to be evaluated. If the condition is true, the loop completes and control passes to the statement immediately after END LOOP.

- ☐ Until the condition is true, the EXIT-WHEN statement acts like a NULL statement, except for evaluating the condition, and does not terminate the loop.
- ☐ A statement inside the loop must change the value of the condition.

Exit and Exit When examples



25

Exit

```
DECLARE
  a number(2) := 10;
BEGIN
  -- while loop execution
  WHILE a < 20 LOOP
    dbms_output.put_line ('value of a: ' || a);
    a := a + 1;
    IF a > 15 THEN
      -- terminate the loop using the exit statement
      EXIT;
    END IF;
  END LOOP;
END;
/
```

Exit When

```
DECLARE
  a number(2) := 10;
BEGIN
  -- while loop execution
  WHILE a < 20 LOOP
    dbms_output.put_line ('value of a: ' || a);
    a := a + 1;
    -- terminate the loop using the exit when
    statement
    EXIT WHEN a > 15;
  END LOOP;
END;
/
```

Continue Control Structure



26

The **CONTINUE** statement causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating. In other words, it forces the next iteration of the loop to take place, skipping any code in between.

Example

```
DECLARE
  a number(2) := 10;
BEGIN
  -- while loop execution
  WHILE a < 20 LOOP
    dbms_output.put_line ('value of a: ' || a);
    a := a + 1;
    IF a = 15 THEN
      -- skip the loop using the CONTINUE statement
      a := a + 1;
      CONTINUE;
    END IF;
  END LOOP;
END;
```

GOTO Control Structure



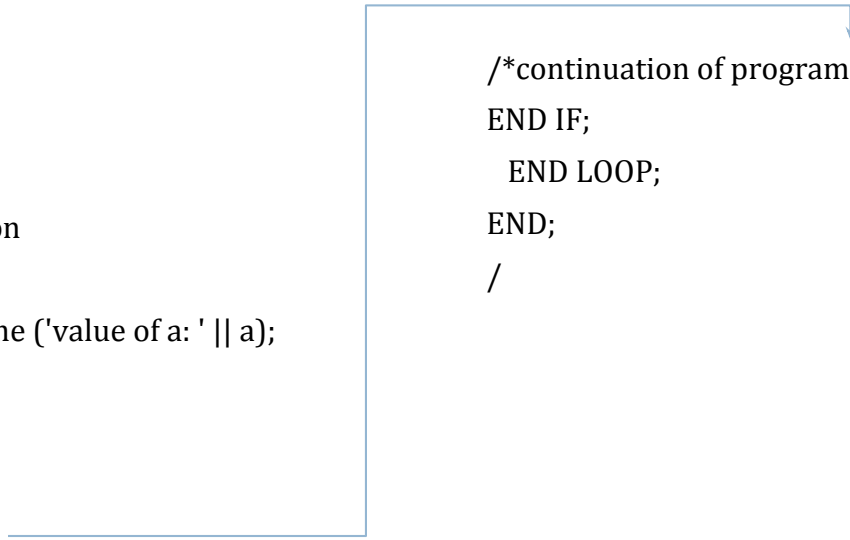
27

A GOTO statement in PL/SQL programming language provides an unconditional jump from the GOTO to a labeled statement in the same subprogram.

NOTE: Use of GOTO statement is highly discouraged in any programming language because it makes difficult to trace the control flow of a program, making the program hard to understand and hard to modify. Any program that uses a GOTO can be rewritten so that it doesn't need the GOTO.

Example

```
DECLARE
  a number(2) := 10;
BEGIN
  <<loopstart>>
  -- while loop execution
  WHILE a < 20 LOOP
    dbms_output.put_line ('value of a: ' || a);
    a := a + 1;
    IF a = 15 THEN
      a := a + 1;
      GOTO loopstart;
    /*continuation of program */
  END IF;
  END LOOP;
END;
/
```

A blue line originates from the 'GOTO loopstart;' statement and extends to the right. It then turns upwards and to the left, ending with a downward-pointing arrowhead that targets the '<<loopstart>>' label at the beginning of the loop.

GOTO Statement Restriction



28

GOTO Statement in PL/SQL imposes the following restrictions:

- ☐ A GOTO statement cannot branch into an IF statement, CASE statement, LOOP statement or sub-block.
- ☐ A GOTO statement cannot branch from one IF statement clause to another or from one CASE statement WHEN clause to another.
- ☐ A GOTO statement cannot branch from an outer block into a sub-block (that is, an inner BEGIN-END block).
- ☐ A GOTO statement cannot branch out of a subprogram. To end a subprogram early, either use the RETURN statement or have GOTO branch to a place right before the end of the subprogram.
- ☐ A GOTO statement cannot branch from an exception handler back into the current BEGIN-END block. However, a GOTO statement can branch from an exception handler into an enclosing block.

Assignment



29

1. Write a PL/SQL program to perform the addition, subtraction, division and multiplication of two user supplied integers and display the result.
2. Write a PL/SQL program to find the square, cube, and double of a input number and print results.
3. Write a PL/SQL program to swap the values of two variables. Print the output before and after swapping.
4. Write a PL/SQL program with two user supplied variables i.e. the first name and the last name. Print the full name with last name and first name separated by comma and a space.
5. Write a PL/SQL program to convert given seconds into its equivalent hours, minutes and seconds. Example. 7560 second = 2 hours, 27 minutes and 40 seconds
6. Write a PL/SQL program to find the average mark of 5 subjects of a student. Assume full mark for each subject is 100.
7. A cashier has currency notes of denominations 10, 50 and 100. If the amount to be withdrawn is input through the keyboard in hundreds, find the total number of currency notes of each denomination the cashier will have to give to the withdrawer.

Assignment cont...



30

8. Write a PL/SQL program to find centigrade for a given Fahrenheit temperature.
9. Write a PL/SQL program to add two times in hour, minute & second format entered through the keyboard in the format hh:mm:ss
10. Ramesh's basic salary is input through the keyboard. His dearness allowance is 40% of basic salary, and house rent allowance is 20% of basic salary. Write a PL/SQL program to calculate his gross salary.
11. Write a PL/SQL program to view the name of the employee and school name whose employee ID is 180750.
12. Write a PL/SQL program to view the name of the employee and job title whose employee ID is 150750.

Assignment



31

Reference Tables

- ❑ **EMPLOYEE** table with the attributes:
ID, LAST_NAME, FIRST_NAME, MIDDLE_NAME, FATHER_NAME,
MOTHER_NAME, SEX, HIRE_DATE, ADDRESS, CITY, STATE, ZIP, PHONE,
PAGER, SUPERVISOR_ID, DOB, INJECTED_DATE
- ❑ **SCHOOL** table with the attributes:
ID, NAME, INJECTED_DATE
- ❑ **EMPLOYEE_ALIGNMENT** table with the attributes:
EMPLOYEE_ID, SCHOOL_ID, INJECTED_DATE
- ❑ **JOB** table with the attributes:
ID, NAME, TITLE, SALARY, BONUS , INJECTED_DATE
- ❑ **EMPLOYEE_PAY** table with the attributes:
EMPLOYEE_ID, JOB_ID, INJECTED_DATE

Assignment



32

13. Write a PL/SQL program to find out if a year is a leap year.
14. Write a PL/SQL program to print all odd numbers between 1 and 10 using a basic loop.
15. Using a FOR loop, print the values 10 to 1 in reverse order.
16. Create a table called ITEM with one column ItemNum with NUMBER type. Write a PL/SQL program to insert values of 1 to 5 for ItemNum using PL/SQL loop.
17. Input a user supplied number and then print its multiplication table (from 1 to 10) using a WHILE loop.
18. Input a month number between 1 and 12 and a 4-digit year, and print number of days in that month.
19. Write a PL/SQL program to test whether the date provided by the user falls on the weekend (in other words, if the day is Saturday or Sunday).
20. Write a PL/SQL program to display the full name, Hire Date, phone number of the employee whose ID is "140329"

Assignment



33

21. Write a PL/SQL program to display the full name, Job Title and Salary of the employee whose ID is "140345"
22. Write a PL/SQL program to display the full name, Job Title, School name and Salary of the employee whose ID is "150985"
23. Write a PL/SQL program to display the full name of the youngest employee in the school of computer engineering
24. Write a PL/SQL program to display the number of years, number of months and days of the oldest employee between hire date and today
25. Write a PL/SQL program to display the average age of the assistant professor in school of mechanical engineering.
26. Write a PL/SQL program to display the job title, salary, bonus, name and employee ID of school of electrical engineering dean.

Yes, I can





Thank You

End of Lab 8