iSQLplus:
http://uadisq01.uad.ac.uk:5560/isqlplus/

Sa0951a
PL/SQL 1: Introduction

An introduction to the procedural language in Oracle

Contents

- What is PL/SQL?
- Purpose – what is it for?
- Block structure
- Anonymous blocks
- Main features
  - rules
  - Variables and data types
  - Loops, branching
- Lots of examples

What is PL/SQL?

- Oracle's **p**rocedural programming **l**anguage extension to SQL
- SQL is embedded in PL/SQL
- Very powerful
  - We're scratching the surface today but hang on as we go deeper over the next few weeks
- We shall be writing code structures called PROCEDURES, FUNCTIONS, TRIGGERS, CURSORS

A very short introduction:

- Variables, constants: Used to temporarily store information
- Loops: tell Oracle to repeat an action – x times, or until some goal has been reached
- Conditional branching: IF statements tell Oracle to do different things depending on some condition
- Functions: stored programs that perform a specific action. E.g. output a value, calculate something

PL/SQL Blocks

- A PL/SQL program is also called a **Block**
- **Anonymous block**:
  - Embedded in application program, stored as script file or typed in directly
  - Not stored by DB
- Named block:
  - Can be stored
  - **Procedures, functions** are examples

Some PL/SQL rules

- Don’t abbreviate keywords
- Put spaces after and between keywords
- Each PL/SQL statement ends with a semi-colon (;)
- SQL takes the same form as outside of PL/SQL
- There is no case-sensitivity (except inside quotes)
- Blocks can be nested up to 200 deep – good luck!
- Good practice to indent nested code though not a syntactical requirement

Block structure

[DECLARE]	declare variables and constants
BEGIN	lists executable statements
[EXCEPTION]	error handling section
END;	ends the block

- Declarations and exceptions are optional
- Need to add a forward slash (/) at end on new line to force execution

Variables and Constants

- Defined in DECLARE statement
  - This creates spaces in memory for temporary storage of data of a specific type
- Constant values are fixed.
- Variables can of course vary during execution!

Defining variables and constants

- Variables
  - Variable_name datatype;
  - Variable_name datatype := expression or value;
  - Variable_name datatype NOT NULL := expression or value;
  - Are set to NULL by default
- Constants
  - constant_name CONSTANT datatype := expression or value;

Block 1_1

Example

```
DECLARE
  v_surname          varchar2(20);
  v_salary           number(9,2):=0;
  v_tax        number(9,2):=ROUND(v_salary*0.25,2);
  v_snum       number(4) NOT NULL :=0;
  c_tax1 CONSTANT number(3,2) := 0.10;
  c_tax2 CONSTANT number(3,2) := 0.23;
BEGIN
  v_snum := v_salary - v_tax;
END;
/
```

- This works but gives no output, neither stores result

Some Rules

- Each statement ends with semi-colon;
- Variable names are not case sensitive and may be up to 30 characters long
- Expressions can contain references to previously defined variables or constants in the current DECLARE section
- Forward references are NOT allowed
- Each variable or constant name in the same block must be unique

Anchoring data types: %TYPE

```
Generally …….
   v_varname                    table.column%type;
   c_constname   CONSTANT       table.column%type :=
   expression;


DECLARE
   v_surname            personnel.SURNAME%TYPE;
   v_bonus              personnel.BONUS%TYPE;
BEGIN
……..................
```

Block 1_2

Displaying output with DBMS_OUTPUT and PUT_LINE

```
SET SERVEROUTPUT ON
DECLARE
  v_surname  varchar2(20):= 'BROWN';
  v_salary         number(9,2) := 10000;
BEGIN
  DBMS_OUTPUT.PUT_LINE('Print these details');
  DBMS_OUTPUT.PUT_LINE('--------------------');
  DBMS_OUTPUT.PUT_LINE(v_surname||' earns '||v_salary);
  DBMS_OUTPUT.PUT_LINE('++++++++++++++++++');
END;
/
```

PL/SQL procedure successfully completed.

The NULL statement

- Performs a null operation (i.e. nothing) and is a useful command to have while developing and testing code (i.e. a placeholder)

```
Begin
...
   IF v_salary > 30000
   THEN NULL;          -- write this later
  END IF;
...
END;
/
```

Block 1_2a

Prompting for a value

```
DECLARE
v_divname      branch.divname%Type;
v_surname      personnel.surname%Type :=
  UPPER('&surname');
BEGIN
  DBMS_OUTPUT.PUT_LINE('converted to: '||v_surname);
END;
/
```

The program will pause when it encounters the “&” character and prompt for a surname

block 1_3

Ok – spot the errors

```
DECLARE
  v_surname          varchar2(10);
  v_N1          number=23.4567;
  v_joindate    date
  v_incep            date:="28-Jan-67";
  v_N2          number:=SQRT(Round(v_N1/3.4,2));
  v_maxbonus    number(3,2):= 300.67;
  v_stockout    bolean:=false;
BEGIN
  NULL; -- develop associated code later
END
/
```

SELECT ….. INTO ……

- One of the key issues of PL/SQL is to extract data from a database to perform some other process

```
SELECT <attribute(s)>
INTO        variable
FROM        <table(s)>
WHERE  <condition>
…
```

Variable MUST be declared prior to use

block 1_4

Example (only works for 1 row!)

```
DECLARE
   v_surname          personnel.surname%type;
   v_bonus            personnel.bonus%type;
BEGIN
   SELECT surname, bonus*1.15
   INTO v_surname, v_bonus
   FROM PERSONNEL
   WHERE SNUM = 3200;
   DBMS_OUTPUT.PUT_LINE(v_surname||' earns ' ||v_bonus);
END;
/
```

Note the single ; at end of SQL code

- Would output - “RAINES earns 575”

Fuller Example

```
DECLARE qty_on_hand NUMBER(6);
BEGIN
  SELECT quantity INTO qty_on_hand FROM inventory
    WHERE product = 'golf club';
  IF qty_on_hand > 0 THEN
    UPDATE inventory SET quantity = quantity - 1
      WHERE product='golf club';
    DBMS_output.put_line('in stock: '||qty_on_hand);
    INSERT INTO purchase_log
      VALUES('Golf club purchased', SYSDATE);
  ELSE
    INSERT INTO purchase_log
      VALUES('out of golf clubs', SYSDATE);
  END IF;
  COMMIT;
END;
/
```

Looks up quantity of golf clubs from inventory table and assigns to variable

Checks > 0

Reduce quantity by 1

Record a message in the purchase log of zero stock

LOOPS

- 3 types:
  - For Loop
  - While Loop
  - Simple or Infinite Loop

FOR Loop

```
BEGIN
  FOR v_count IN 1..10 LOOP
   Insert into test(id_no)
       values(v_count);
  END LOOP;
END;
/
```

TEST

ID_NO
1
2
3
4
5
6
7
8
9
10

Notes:

- v_count is NOT declared -- it is implicit
- The table test with column id_no must exist

While Loop

Block 1_5
To test:
Select * from Test
Run program
Select * from Test

```
DECLARE
 V_count number(2):=1;
BEGIN
 WHILE v_count < 11 LOOP
   Insert into test(id_no)
       values(v_count);
   v_count:=v_count+1;
 END LOOP;
END;
/
```

TEST

ID_NO
1
2
3
4
5
6
7
8
9
10

Infinite Loop Example

```
DECLARE
V_count number(2):=1;
BEGIN
  LOOP
    Insert into test(id_no)
       values(v_count);
    EXIT WHEN v_count=10;
    v_count:=v_count+1;
  END LOOP;
END;
/
```

TEST

ID_NO
1
2
3
4
5
6
7
8
9
10

IF…THEN…ELSIF…ELSE

```
IF condition THEN statement(s);
   [ELSIF condition THEN statement(s);]
[ELSE statement(s);]
END  IF;
```

Note: Yes, it is ELSIF, not ELSEIF and not ELSE IF

If Example

Block 1_6
To test:
Select * from Test
Run block
Select * from Test

```
DECLARE
v_count number(2):=1;
BEGIN
LOOP
    IF v_count between 1 and 5 THEN
    Insert into test values (v_count, 'Group1');
    ELSIF   v_count between 6 and 10 THEN
    Insert into test values (v_count, 'Group2');
    ELSE
    Insert into test values (v_count, 'Group3');
    END IF;
    EXIT WHEN v_count =15;
    v_count:=v_count+1;
END LOOP;
END;
/
```

TEST

ID_NO	DETAILS
1	GROUP1
2	GROUP1
3	GROup1
4	GROUP1
5	GROUP1
6	GROUP2
7	GROUP2
9	

Summary and look Ahead

- Looked at
  - Classical programming structures:
    Loops, Ifs etc
  - Anchoring data types
  - Block structure:
    Declare …. Begin…… Exception … End;
  - Variables and constants
- Going to look at
  - Exception handling, cursors, procedures, functions, triggers, packages hopefully!

Some other useful examples….. <<Labels>> (useful when nesting)

```
BEGIN
<<firstloop>>
FOR counter in 1..2 Loop
    DBMS_OUTPUT.PUT_LINE('1st: '||firstloop.counter);
    <<secondloop>>
    FOR counter in 1..4 Loop
            DBMS_OUTPUT.PUT_LINE('1st: '|| firstloop.counter);
            DBMS_OUTPUT.PUT_LINE('2nd: '|| secondloop.counter);
    END LOOP secondloop;     -- aids readability
    DBMS_OUTPUT.PUT_LINE('--------------------');
END LOOP firstloop;                          -- aids readability
END;
/
```

IF: Conditional Tests supported

- Logicals:
  - AND OR NOT
- Expressions:
  - IS [NOT] NULL, [NOT] BETWEEN a AND b
  - [NOT] like a, [NOT] IN *list*
- Comparisons: < > <= >= <> !=
- Operations: + - * / (and more)
- Functions: any legal SQL function

Reading & References

- Connolly/Begg (3rd/4th ed) Section 8.2.5
- **Oracle® PL/SQL™ by Example** – in Safari e-books
- Morris-Murphy chapters 15 & 16
- Shah, N. (2002). Database Systems Using Oracle. A simplified guide to SQL and PL/SQL. Chapter 8 onwards
- Morrison/Morrison Chapter 4.