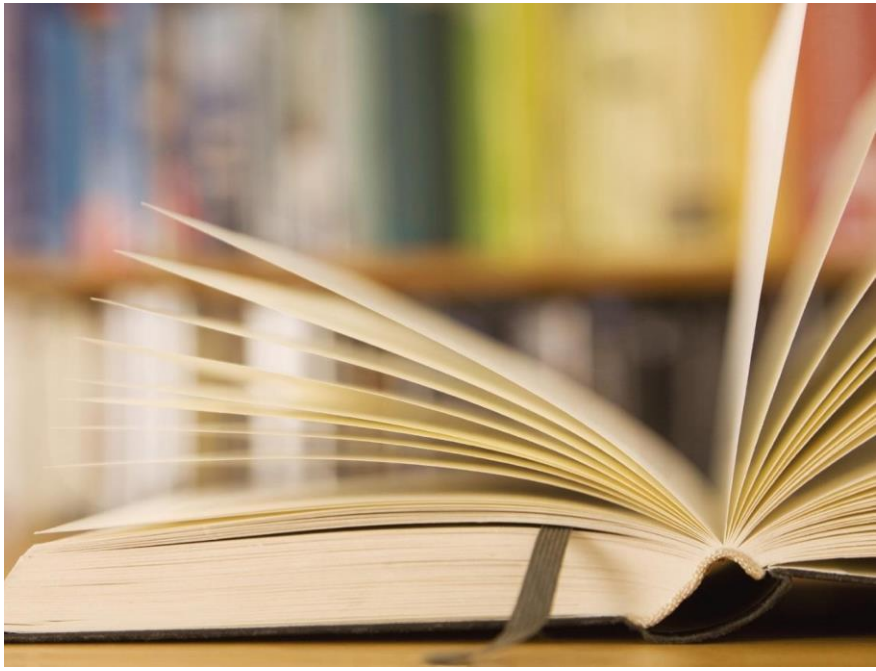
CSCI235 – Database Systems

PL/SQL

sjapit@uow.edu.au

2 October 2021



Acknowledgements

The following presentation were adapted from the lecture slides of:

CSCI235 – Database Systems, 07PLSQL

By Dr Janusz R. Getta, University of Wollongong, Australia

Spring 2018

Outline

- PL/SQL? What is it? Why do we need it?
- Program structure
- Declarative, Executable, Exception components
- Structures of anonymous blocks, procedures, and functions
- Data types, implicit type declarations
- Operators
- Control statements
- Cursors
- Exceptions

PL/SQL

PL/SQL, What is it?
Why do we need it?

PL/SQL? What is it? Why do we need it?

- PL/SQL is a procedural extension of **SQL**
- **PL/SQL** = procedural **P**rogramming **L**anguage + **SQL**
- We need PL/SQL to bridge a gap between a high level declarative query language and a procedural programming language
- PL/SQL is a subset of a programming language **Ada**

PL/SQL? What is it? Why do we need it?

PL/SQL =

- Data Manipulation statements of SQL +
- SELECT statement +
- Variables +
- Assignment statement +
- Conditional control statements +
- Repetition statement +
- Exception handling +
- Procedure and function statements +
- packages

PL/SQL

Program Structure

Program Structure

- PL/SQL is a block-structured language
- It means that its basic units such as anonymous blocks, procedures, and functions are the logical blocks.
- Anonymous block is persistent for only a single processing, that it, it is not stored in a data dictionary.
- A named block, either procedure or function, is persistent for many processing, that is, it can be stored in a data dictionary.
- Logical blocks can be nested to any level

Program Structure

- Logical blocks consists of declarative, executable, and exception components
- A declarative component consists of declarations of constants, variables, types, methods, cursors, etc., and it is optional
- An executable component consists of executable code and must have at least one statement.
- An exception component consists of executable code handling exceptions and it is optional

Program Structure

- A sample anonymous block

```
-- A sample single line comment
DECLARE    -- A keyword, beginning of declarative component

/* Declarative
        component        a sample multiline comment                */
BEGIN        -- A keyword, beginning of executable component
/* Executable
        component                                                    */
```

Program Structure

A sample anonymous block

```
-- A sample single line comment
DECLARE   -- A keyword, beginning of declarative component

/* Declarative
        component    a sample multiline comment                */

BEGIN       -- A keyword, beginning of executable component

/* Executable
        component                                                 */
```

Program Structure

A sample anonymous block (…continue)

```
NULL;           -- it must include at least one statement,
                -- NULL; is an optional empty statement

EXCEPTION       -- A keyword, beginning of exception component
/* Exception
        component                                          */

END;            -- A keyword, end of anonymous block
/               -- A forward slash line means; execute this
                -- procedure
```

Program Structure

```
DECLARE
    Definition of PL/SQL objects to be used within
    this block
BEGIN
    Executable actions
EXCEPTION
    Exception Handlers – what to do if the
    executable actions  cause an error condition
END;
/
```

PL/SQL

Declarative, Executable, Exception components

Declarative Components

Declarative components contain declarations of variables, constants, cursors, procedures, and functions

```
DECLARE
    stock_num          NUMBER(5);
    stock_name         VARCHAR2(30);
    stock_date         DATE;
    stock_required     NUMBER(5) := 30;
    limit CONSTANT     NUMBER(11,2) := 2.45;
```

Declarative Components

```
stock_value STOCK.value%TYPE;
stock_row STOCK%ROWTYPE;
CURSOR Q IS
    SELECT snum
    FROM STUDENT
    WHERE name = 'Jo';
```

Executable Components

Executable components include assignment statements, conditional control statements, iterative statements, procedure and function calls, SQL statements

```
student_num := 910000;
SELECT      name
INTO        student_name
FROM        STUDENT
WHERE       s#=student_num;
```

Executable Components

```
IF (a > b)
THEN
    a := a + 1;
    c := c + 2;
ELSEIF (a < b)
    c := c - 2;
ELSE
    b := b + 1;
END IF;
```

Executable Components

```
FOR i IN 1..100 LOOP
    b := b – i;
END LOOP;
```

Exception Components

Exception component consists of executable statements that service the exceptional situations during execution

```
EXCEPTION
WHEN NO_DATA_FOUND THEN
    INSERT INTO AUDIT_TABLE VALUES( SYSDATE, snum )
WHEN OTHERS
    i := i + 1;
    UPDATE DEPARTMENT
        SET budget := i * budget;
END;
```

Exception Components

```
DECLARE
    too_large EXCEPTION;
BEGIN
    IF a > 100000 THEN
        RAISE too_large;
    END IF;
EXCEPTION
    WHEN too_large;
        DBMS_OUTPUT.PUTLINE ('Too large!');
END
```

PL/SQL

Structures of anonymous blocks, procedures, and functions

Structure of anonymous block

A bird-eye view of an anonymous block is the following:

```
DECLARE
    -- Optional declarations
BEGIN
    -- executable statements, at least one statement is required
EXCEPTION
    -- optional exception handlers
END;
/   -- processing command
```

Structure of anonymous block

A sample Hello world! anonymous block

```
SET SERVEROUTPUT ON
BEGIN
        DBMS_OUTPUT.PUT_LINE ('Hello world!');
END;
/
```

Structure of anonymous block

Processing SQL statements in a sample anonymous block

```
DECLARE
    average NUMBER(8,2)
BEGIN
    SELECT avg(budget) INTO average
    FROM  DEPARTMENT;
    IF average < 3000 THEN
       UPDATE DEPARTMENT
       SET budget := budget + 100;
    END IF
END;
/
```

Structure of Procedure

A birds-eye view of a procedure is the following

```
PROCEDURE procedure_name (parameters) IS
    -- optional declarations
BEGIN
    -- executable statements, at least one statements is required
EXCEPTION
    -- optional exception handlers
END procedure_name;
```

Structure of Procedure

A sample hello world procedure

```
PROCEDURE hello_world ( hello IN VARCHAR2,
                        world IN VARCHAR2,
                        hello_word OUT VARCHAR2) IS
BEGIN
   hello_world := hello || ' ' || world || '!';
END hello_world;
```

A Sample Procedure

Processing SQL statements in a sample procedure

```
PROCEDURE raise_budget ( department_name IN VARCHAR2,
                                    budget_limit IN NUMBER) IS
 Current_budget DEPARTMENT.budget%TYPE;
 BEGIN
     SELECT  budget INTO current_budget
     FROM    DEPARTMENT
     WHERE name = department_name;
```

A Sample Procedure

```
    IF current_budget < budget_limit THEN
        UPDATE DEPARTMENT
        SET budget := budget_limit
        WHERE name = department_name;
    ELSE
        INSERT INTO AUDIT VALUES ('Math budget OK',
        current_gudget);
    END IF;
    COMMIT;
END raise_budget;
```

Structure of Functions

A birds-eye view of a function is the following

```
FUNCTION function_name ( parameters )
RETURN type-specification IS
    -- optional declarations
BEGIN
    -- executable statements, at least one statements is required
EXCEPTION
    -- optional exception handlers
END function_name
```

Structure of Functions

Sample hello world function

```
FUNCTION hello_world ( hello IN VARCHAR2,
                       world IN VARCHAR2 ) IS
RETURN VARCHAR2 IS
BEGIN
   RETURN hello || ' ' || world || '!';
END hello_world;
```

Structure of Functions

Processing SQL statements in a sample function

```
FUNCTION raise_budget (
                        department_name IN VARCHAR2,
                        budget_limit        IN NUMBER )
RETURN NUMBER IS
    current_budget DEPARTMENT.budget%TYPE;
BEGIN
    SELECT  budget INTO current_budget
    FROM    DEPARTMENT
    WHERE  name = department_name;
```

Structure of Functions

```
    IF current_budget < budget_limit THEN
        UPDATE   DEPARTMENT
        SET      budget := budget_limit
        WHERE    name = department_name;
        RETURN budget_limit;
    ELSE
        INSERT INTO AUDIT VALUES ('Math budget Ok',
        current_budget);
        RETURN current_budget;
    END IF;
    COMMIT;
END raise_budget;
```

PL/SQL

Data types, implicit type declarations

Data Types

- Some of the predefined data types in PL/SQL

INTEGER, DECIMAL, NUMBER, CHAR, DATE, VARCHAR, VARCHAR2, LONG, BOOLEAN, ROWID, EXCEPTION

Sample implicit type declarations

```
DECLARE
    student_no      STUDENT.snum%TYPE;
    student_name    STUDENT.name%TYPE;

    student_row     STUDENT%ROWTYPE;
```

Data Types

```
BEGIN
    student_no := 1234567;
    SELECT      name INTO student_name
    FROM        STUDENT
    WHERE       snum = student_no;
    student_row.snum := 1234567;
    student_row.name := 'James';
    student_row.dob := TO_DATE('24-SEP-2021', 'DD-MON-YYYY');
    INSERT INTO STUDENT VALUES (student_row.snum,
    student_row.name, student_row.dob);
END;
```

PL/SQL

Operators

Operators

- Arithmetic operators

  +, -, *, /, **

- Relational operators

  <, >, >=, <=, =, !=, <>, -=

- Comparison operators

  LIKE, BETWEEN, IN, IS NULL, =, != <>, ~=

Operators

- Boolean operators

  AND, OR, NOT

- String operators

  ||

- Operator precedence

  (**), (unary +, -), (*, /), (+, -, ||), (comparison), (NOT), (AND), (OR)

PL/SQL

Control statements

Conditional control statements

A birds-eye view of conditional control statements is the following

```
IF condition THEN
        statement;
        …
ELSE
        statement;
        …
END IF;
```

```
IF condition THEN
        statement;
        …
ELSIF condition THEN
        statement;
        …
ELSIF condition THEN
        statement;
        …
ELSE
        statement;
        …
END IF;
```

Iterative control statements

A birds-eye view of iterative control statements is the following

```
LOOP
        statement;
        …
        IF condition THEN
            EXIT;  statement;
            …
        END IF;
        statement;
        …
END LOOP;
```

```
FOR variable IN
scope  LOOP
    statement;
    …
END LOOP;
FOR variable IN REVERSE
scope  LOOP
    statement;
    …
END LOOP;
```

Iterative control statements

A birds-eye view of iterative control statements is the following

```
WHILE (condition)
LOOP
        statement
        …  END
LOOP;
```

```
LOOP
    statement;
    …
    EXIT WHEN condition;
    statement;
    …
END LOOP;
```

PL/SQL

Cursors

Cursors

What happens when SELECT statement returns more than one row?

```
DECLARE
    student_no      STUDENT.snum%TYPE;
BEGIN
    SELECT      snum INTO student_no
    FROM        STUDENT
    WHERE       name = 'Pam';
    …
```

ERROR at line 1:
ORA-06503: PL/SQL: error 0 - Unhandled exception ORA-01427: single-row subquery returns more than one row which was raised in a statement ending at line 6

Cursors

- A variable student_no cannot be used to store several rows retrieved from a relational table
- A solution is to process the rows in a *row by row* mode
- A cursor is a construction that allows for processing the rows retrieved from the relational tables in a *row by row* mode.

- Cursor can be implemented as
  - Explicit cursor
  - Implicit cursor

Cursors

- Explicit declaration and processing of a cursor

```
DECLARE
   student_no STUDENT.snum%TYPE;
```

```
CURSOR Q IS
   SELECT snum
   FROM
   STUDENT
   WHERE name = 'Pam';
```

Cursors

```
BEGIN
    OPEN Q;
    LOOP
        FETCH Q INTO student_no;
        IF Q%NOTFOUND THEN
            EXIT;
        END IF;
        INSERT INTO PAM VALUES(student_no)
    END LOOP;
    CLOSE Q;
    COMMIT;
END;
```

Implicit cursor processing

Implicitly declaration and processing of a cursor

```
BEGIN
FOR Q_row IN (SELECT snum
                    FROM STUDENT
                    WHERE name = 'Pam')
LOOP
      INSERT INTO PAM VALUES(Q_row.snum);  END
LOOP;
COMMIT;
END;
```

Implicit Cursor Processing

- A cursor is implicitly declared
- A cursor is implicitly opened
- A row is implicitly fetched
- End of table condition is implicitly checked
- A cursor is implicitly closed

Cursor Attributes

- A cursor attribute determines a state of a cursor.
- A cursor attribute %NOTFOUND evaluates to true if the last FETCH failed because no more rows were available.
- A cursor attribute %FOUND evaluates to true if the last FETCH succeeded.
- A cursor attribute %ROWCOUNT evaluates to the total number of rows FETCHED so far.
- A cursor attribute %ISOPEN evaluates to true if a cursor is OPENed.
- You can find more information about cursor attributes here.

Cursor Attributes

A sample testing of cursor attributes

```
DECLARE
        student_no      STUDENT.snum%TYPE;
        CURSOR Q IS
           SELECT  snum
           FROM    STUDENT
           WHERE NAME = 'Pam';
        BEGIN
           OPEN Q;
```

Cursor Attributtes

A sample testing of cursor attributes (… continue)

```
LOOP
    FETCH Q INTO student_no;
    IF Q%NOTFOUND THEN
        EXIT
    END IF;
    INSERT INTO PAM VALUES(student_no);
END LOOP;
```

Cursor Attributes

A sample testing of cursor attributes (… continue)

```
    IF Q%ROWCOUNT = 0 THEN
        INSERT INTO MESSAGES VALUES ('NO ROWS
        PROCESSED');
    END IF;
    CLOSE Q;
    COMMIT;
END;
```

PL/SQL

Exceptions

Exceptions

- An **exception** is an internally defined or user defined error condition, e.g. divide by zero, no rows selected by **SELECT** statement with **INTO** clause, failure of **FETCH** statement, use of a cursor which has not been opened yet, etc.
- A typical exception handling

```
DECLARE
    error_number NUMBER(5);
    error_message VARCHAR(200);
    …
```

Exceptions

```
EXCEPTION
   WHEN OTHERS THEN
      error_number = SQLCODE;
      error_message = SQLERRM;
      DBMS_OUTPUT.PUT_LINE(error_number || '-' ||
      error_message);
      INSERT INTO ERRORS (error_number,
      error_message);
      COMMIT;
END;
```

Exceptions

- Handling an empty answer (respond) from SELECT statement

```
DECLARE
   student_name STUDENT.name%TYPE;
BEGIN
   SELECT name INTO student_name
   FROM  STUDENT
   WHERE snum = 1234567;
    …
```

Exceptions

```
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        INSERT INTO MESSAGES VALUES('Student not found');
        COMMIT;
END;
```

Exceptions

- An **exception NO_DATA_FOUND** is raised when **SELECT** statement returns no rows
- An **exception TOO_MANY_ROWS** is raised when **SELECT** statement returns more than one row
- An **exception INVALID_CURSOR** is raised when PL/SQL call specifies an invalid cursor,e.g. closing an unopened cursor
- An **exception OTHERS** is raised when any other exception, not explicitly named happens
- You can find a complete list of PL/SQL **exception**s here

References

- Database PL/SQL Language Reference

- T. Connoly, C. Begg, Database Systems, A Practical Approach to Design, Implementation, and Management, Chapter 8 Advanced SQL, Pearson Education Ltd, 2015