CSCI235 Database Systems

PL/SQL

Dr Janusz R. Getta

School of Computing and Information Technology -
University of Wollongong

PL/SQL

Outline

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

PL/SQL is a procedural extension of SQL

PL/SQL = procedural Programming Language + 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 =

- Data Manipulation statements of SQL +
- `SELECT` statement +
- variables +
- assignment statement +
- conditional control statements +
- repetition statement +
- exception handling +
- procedure and function statements + packages

PL/SQL

Outline

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, i.e. it is not stored in a data dictionary

A named block (either procedure or function) is persistent for many processings, i.e. it can be stored in a data dictionary

Logical blocks can be nested to any level

Logical blocks consist 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

Program structure

An exception component consists of executable code handling exceptions and it is optional

A sample anonymous block

PL/SQL
```
-- A sample single line comment
DECLARE                 -- A keyword, beginning of declarative component
/*      Declarative
         component         A sample multiline comment                        */
```

PL/SQL
```
BEGIN                   -- A keyword, the beginning of executable component
/*      Executable component                                                  */
NULL;                   -- it must include at least one statement,
                        -- NULL; is an optional empty statement
```

PL/SQL
```
EXCEPTION               -- A keyword, the beginning of exception component
/*      Exception component                                                           */
END;                    -- A keyword, the end of anonymous block
/                       -- A forwad slash line means: execute this procedure
```

PL/SQL

Outline

Declarative components

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

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

```
SELECT name                                                           PL/SQL
INTO student_name
FROM STUDENT
WHERE s# = student_num;
```

```
IF (a > b)                                                            PL/SQL
THEN
  a := a + 1;
  c := c + 2
ELSIF (a < b)
  c := c - 2
ELSE
  b:= b + 1
END IF;
```

```
FOR i IN 1..100 LOOP                                                  PL/SQL
  b := b - i
END LOOP;
```

Exception components

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

PL/SQL
```
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;
```

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

PL/SQL

Outline

Structure of anonymous block

A birds-eye view of an anonymous block is the following

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

A sample `Hello world !` anonymous block

```
SET SERVEROUTOUT ON                                              SQL*Plus
```

```
BEGIN                                                            PL/SQL
  DBMS_OUTPUT.PUT_LINE('Hello world !');
END;
/
```

A sample anonymous block

Processing SQL statements in a sample anonymous block

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

Structure of procedure

A birds-eye view of a procedure is the following

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

A sample `hello world` procedure

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

A sample procedure

Processing SQL statements in a sample procedure

```
PROCEDURE raise_budget(                                                        PL/SQL
                        department_name IN VARCHAR,                            PL/SQL
                        budget_ limit IN NUMBER ) IS                           PL/SQL
  current_budget DEPARTMENT.budget%TYPE;                                       PL/SQL
BEGIN                                                                          PL/SQL
                                                                               PL/SQL
  SELECT budget INTO current_budget  FROM DEPARTMENT  WHERE name = department_name;
  IF current_budget < budget_limit THEN                                        PL/SQL
    UPDATE DEPARTMENT SET budget := budget_limit WHERE name = department_name; PL/SQL
  ELSE                                                                         PL/SQL
    INSERT INTO AUDIT VALUES( 'Math budget OK', current_budget);               PL/SQL
  END IF;                                                                      PL/SQL
  COMMIT;                                                                      PL/SQL
END raise_budget;                                                              PL/SQL
```

Structure of function

A birds-eye view of a function is the following

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

A sample `hello world` function

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

Structure of function

Processing SQL statements in a sample function

```
FUNCTION raise_budget(
                       department_name IN VARCHAR,
                       budget_ limit IN NUMBER )
RETURN NUMBER IS
  current_budget DEPARTMENT.budget%TYPE;
BEGIN

  SELECT budget INTO current_budget FROM DEPARTMENT WHERE name = department_name;
  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

Outline

Data types

Some of the predefined data types in PL/SQL

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

PL/SQL

Sample implicit type declarations

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

PL/SQL

```
  student_row STUDENT%ROWTYPE;
```

PL/SQL

```
BEGIN
  student_no := 1234567;
```

PL/SQL

```
  SELECT name FROM STUDENT INTO student_name WHERE snum = student_no;
```

PL/SQL

```
  student_row.snum := 1234567;
  student_row.name := 'James';
  student_rec.dob := TO_DATE('01-DEC-1994', 'DD-MON-YYYY');
  INSERT INTO STUDENT VALUES(student_row.snum, student_row.name, student_row.dob);
```

PL/SQL

PL/SQL

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

Operators

Arithmetic operators

```
+, -, *, /, **
```
PL/SQL

Relational operators

```
<, >, >=, <=, =, !=, <>, ~=
```
PL/SQL

Comparison operators

```
LIKE, BETWEEN, IN, IS NULL, =, !=, <>, ~=
```
PL/SQL

Boolean operators

```
AND, OR, NOT
```
PL/SQL

String operator

```
||
```
PL/SQL

Operator precedence

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

PL/SQL

Outline

Conditional control statements

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

```
IF condition THEN                                                      PL/SQL
  statement;
  ...
ELSE
  statement;
  ...
END IF;
```

```
IF condition THEN                                                      PL/SQL
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                                                                PL/SQL
  statement;
  ...
  IF condition THEN EXIT;
    statement;
    ...
  END IF;
  statement;
  ...
END LOOP;
```

```
 FOR variable IN scope                                              PL/SQL
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)                                    PL/SQL
LOOP
  statement;
  ...
END LOOP;
```

```
LOOP                                                 PL/SQL
  statement;
  ...
   EXIT WHEN condition;
  statement;
  ...
END LOOP;
```

PL/SQL

Outline

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
```

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

Cursors

Explicit declaration and processing of a cursor

PL/SQL
```
DECLARE
  student_no STUDENT.snum%TYPE;
```

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

PL/SQL
```
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;
```

PL/SQL

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 `FETCH`ed 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

PL/SQL
```
DECLARE
  student_no STUDENT.snum%TYPE;
  CURSOR Q IS
    SELECT snum FROM STUDENT WHERE name = 'Pam';
BEGIN
  OPEN Q;
  LOOP
    FETCH Q INTO student_no;
    IF Q%NOTFOUND THEN
      EXIT
    END IF;
    INSERT INTO PAM VALUES(student_no);
  END LOOP;
```

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

PL/SQL

Outline

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

PL/SQL
```
DECLARE
  error_number NUMBER(5);
  error_message VARCHAR(200);
  ...
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 empty an answer from `SELECT` statement

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

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

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 exceptions 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