PL/SQL TUTORIAL

April
10, 2012

Doan Quang Minh

CTI Cosatech V

About this slide

This Oracle PL/SQL tutorial guides you the basics of programming in PL/SQL with appropriate examples. You can use this tutorial as your reference while programming with PL/SQL.

You should have basic knowledge about SQL and database management before reading this.

Agenda

I. Introduction (5 minutes)
II. Basic syntax (15 minutes)
III. Advanced syntax (30 minutes)
IV. Conclusion (2 minutes)
V. Reference (2 minutes)
VI. Keywords (3 minutes)
VII. Q&A

I. Introduction

1. Introduction to PL/SQL
   - What is PL/SQL?
   - The PL/SQL Engine
2. Advantages of PL/SQL
   - Block Structures
   - Procedural Language Capability
   - Better Performance
   - Error Handling

1. Introduction to PL/SQL

What is PL/SQL?

- Procedural Language extension of SQL.
- PL/SQL is a combination of SQL along with the procedural features of programming languages. It was developed by Oracle Corporation in the early 90's to enhance the capabilities of SQL.

1. Introduction to PL/SQL(cont)

The PL/SQL Engine

- Oracle uses a PL/SQL engine to processes the PL/SQL statements.
- A PL/SQL code can be stored in the client system (client-side) or in the database (server-side).

2. Advantages of PL/SQL

- **Block Structures**: PL SQL consists of blocks of code, which can be nested within each other. Each block forms a unit of a task or a logical module. PL/SQL Blocks can be stored in the database and reused.

- **Procedural Language Capability**: PL SQL consists of procedural language constructs such as conditional statements (if else statements) and loops like (FOR loops).

2. Advantages of PL/SQL(cont)

- **Better Performance**: PL SQL engine processes multiple SQL statements simultaneously as a single block, thereby reducing network traffic.

- **Error Handling**: PL/SQL handles errors or exceptions effectively during the execution of a PL/SQL program. Once an exception is caught, specific actions can be taken depending upon the type of the exception or it can be displayed to the user with a message.

II. Basic syntax

1. Block Structures
2. Operators
3. Comments
4. Delimiters
5. Variables
6. Constants
7. Records
8. Conditional Statements
9. Iterative Statements

1. Block Structures

- Basic block

```
[DECLARE]
   Variable declaration
BEGIN
   Program Execution
[EXCEPTION]
   Exception handling
END;
```

- Example

```
SET SERVEROUTPUT ON SIZE 1000000
BEGIN
    dbms_output.put_line('Hello PL/SQL');
END;
```

2. Operators

- Comparison operators

```
NOT   IS NULL   LIKE   BETWEEN   IN   AND   OR
+   -   *   /   @   ;   =   <>   !=   ||   <=   >=
```

- Assignment operator

`:=` (You can assign values to a variable, literal value, or function call but NOT a table column)

3. Comments

- Comment

```
-- comment
/* comment */
```

- Example

```
DECLARE
        /* Multi-line comments are not required to actually use
        multiple lines. */
BEGIN
        -- This is a single line comment
        NULL;
END;
```

4. Delimiters

- Delimiter
  - Item separator .
  - Character string delimiter '
  - Quoted String delimiter "
  - Bind variable indicator :
  - Attribute indicator %
  - Statement terminator ;
- Example

```
job_record.jobname := 'Test Job';
v_empno emp.empno%TYPE := &empno;
```

5. Variables

- Variable

```
variable_name    datatype [NOT NULL := value ];
```

- Example

```
DECLARE
        v_first_name     VARCHAR2(20);
        v_employee_id    NUMBER NOT NULL;
        v_last_name      EMPLOYEES.LAST_NAME%TYPE;
        v_employee       EMPLOYEES%ROWTYPE;
        v_hire_date      DATE;
BEGIN
        NULL;
END;
```

6. Constants

- Constant

```
constant_name CONSTANT datatype := VALUE;
```

- Example

```
DECLARE
        c_boss CONSTANT VARCHAR2(4) := 'BOSS';
BEGIN
        NULL;
END;
```

7. Records

- Record

```
TYPE record_type_name IS RECORD(
    first_col_name   column_datatype,
    second_col_name  column_datatype, ...);
record_name          record_type_name;
```

- Example

```
DECLARE
        TYPE employee IS RECORD(
            v_employee_id    NUMBER NOT NULL
          ,v_first_name      VARCHAR2(20));
        v_employee           employee;
BEGIN
        NULL;
END;
```

8. Conditional Statements

- If statement

```
IF condition THEN
    statement 1;
ELSE
    statement 2;
END IF;
```

- Example

```
IF v_employee.sal > 0 THEN
    v_employee.sal := 1;
ELSE
    v_employee.sal := 0;
END IF;
```

9. Iterative Statements

- Iterative statement

```
LOOP
    statements;
    EXIT WHEN condition; (EXIT;)
END LOOP;

WHILE <condition>
    LOOP statements;
END LOOP;

FOR counter IN val1..val2
    LOOP statements;
END LOOP;
```

- Example

```
LOOP
    monthly_val := daily_val * 31;
    EXIT WHEN monthly_value > 4000;
END LOOP;

WHILE monthly_val <= 4000
    LOOP monthly_val := daily_val * 31;
END LOOP;

FOR counter IN 1..9
    LOOP monthly_val := daily_val * 31;
END LOOP;
```

III. Advanced syntax

1. Cursors
2. Explicit Cursors
3. Procedures
4. Functions
5. Parameters-Procedure, Function
6. Exception Handling
7. Triggers

1. Cursors

- What is cursor ?
  - In memory work area
  - Store rows selected from DB
  - Process one row per time only
  - Active set is the set of rows the cursor hold.
- Supported attributes
  - %FOUND
  - %NOTFOUND
  - %ROWCOUNT
  - %ISOPEN

1. Cursors(cont)

- Two types of cursors in PL/SQL
  - Implicit cursors
  - Explicit cursors

- Implicit cursors

  These are created by default when DML statements like, INSERT, UPDATE, and DELETE statements are executed. They are also created when a SELECT statement that returns just one row is executed.

1. Cursors(cont)

- Example

```
DECLARE
   v_rows number(5);
BEGIN
   UPDATE     employee
   SET        salary = salary + 1000;
   IF SQL%NOTFOUND THEN
      dbms_output.put_line('None of the salaries where updated');
   ELSIF SQL%FOUND THEN
      v_rows := SQL%ROWCOUNT;
      dbms_output.put_line('Salaries for ' || v_rows || 'employees are
   updated');
   END IF;
END;
```

2. Explicit Cursors

- What is explicit cursor ?

  An explicit cursor is defined in the declaration section of the PL/SQL Block. It is created on a SELECT Statement which returns more than one row. We can provide a suitable name for the cursor.

- How to use explicit cursors ?
  - **DECLARE** the cursor in the declaration section.
  - **OPEN** the cursor in the Execution Section.
  - **FETCH** the data from cursor into PL/SQL variables or records in the Execution Section.
  - **CLOSE** the cursor in the Execution Section before you end the PL/SQL Block.

2. Explicit Cursors

- Example
  - Declaring a cursor

```
DECLARE
   CURSOR emp_cur IS
   SELECT *
   FROM emp_tbl
   WHERE salary > 5000;
```

  - Accessing the cursor

```
BEGIN
   OPEN emp_cur;
   FETCH emp_cur INTO v_record;
     process_one_record(v_record);
   CLOSE emp_cur;
END;
```

3. Procedures

- What is procedures ?
  - A named PL/SQL block
  - A procedure has a header and a body
  - May or may not return values
  - If stored on DBMS, we call it stored procedures

3. Procedures(cont)

- Declaration

```
CREATE [OR REPLACE] PROCEDURE proc_name [list of parameters]
IS
    Declaration section
BEGIN
    Execution section
EXCEPTION
    Exception section
END;
```

- Execution
  - From the SQL prompt : `EXECUTE [or EXEC] procedure_name;`
  - Within another procedure: `procedure_name;`

3. Procedures(cont)

- Example

```
CREATE OR REPLACE PROCEDURE employer_details IS
      CURSOR emp_cur IS
      SELECT first_name, last_name, salary
      FROM emp_tbl;
      emp_rec emp_cur%ROWTYPE;
    BEGIN
      FOR emp_rec IN sales_cur
      LOOP
          dbms_output.put_line(emp_cur.first_name);
      END LOOP;
    END;

 EXECUTE employer_details;
```

4. Functions

- What is functions?
  - A named PL/SQL block
  - A function has a header and a body
  - Must always return a value (different to a procedure)
  - If stored on DBMS, we call it stored functions

4. Functions(cont)

- Declaration

```
CREATE [OR REPLACE] FUNCTION function_name [parameters]
RETURN return_datatype;
IS Declaration_section
BEGIN
    Execution_section
    Return return_variable;
EXCEPTION
    exception section
    Return return_variable;
END;
```

4. Functions(cont)

- Execution
  - Since a function returns a value we can assign it to a variable

```
variable_name := function_name;
```

  - As a part of a SELECT statement

```
SELECT function_name FROM table;
```

  - In a PL/SQL Statements like,

```
dbms_output.put_line(function_name);
```

4. Functions(cont)

- Example

```
CREATE OR REPLACE FUNCTION employer_details_func
RETURN VARCHAR(20);
IS
   emp_name VARCHAR(20);
BEGIN
   SELECT first_name INTO emp_name
   FROM emp_tbl
   WHERE empID = '100';
   RETURN emp_name;
END;

SELECT employer_details_func FROM dual;
```

5. Parameters in Procedure, Functions

- 3 ways to pass parameters
  - IN-parameters
  - OUT-parameters
  - IN OUT-parameters

NOTE: If a parameter is not explicitly defined a parameter type, then by default it is an IN type parameter.

5. Parameters in Procedure, Functions

- Example

```
CREATE OR REPLACE PROCEDURE emp_name (
        p_id IN NUMBER
      , p_emp_name OUT VARCHAR2(20))
IS
BEGIN
   SELECT first_name INTO p_emp_name
   FROM emp_tbl WHERE empID = p_id;
END;
```

6. Exception Handling

- What is a exception handling?
  - A feature to handle the Exceptions which occur in a PL/SQL Block
  - Avoid the source code from exiting abruptly
  - When an exception occurs a messages which explains its cause is received

- 3 parts of an PL/SQL exception?
  - Type of Exception
  - An Error Code
  - A message

6. Exception Handling

- Structure of Exception Handling

```
EXCEPTION
   WHEN ex_name1 THEN
         -- Error handling statements
   WHEN ex_name2 THEN
         -- Error handling statements
   WHEN Others THEN
         -- Error handling statements
```

6. Exception Handling

- 3 types of exception
  - Named System Exceptions
  - Unnamed System Exceptions
  - User-defined Exceptions

6. Exception Handling

- Example

```
BEGIN
    -- Execution section
EXCEPTION
    WHEN NO_DATA_FOUND THEN
    dbms_output.put_line ('A SELECT...INTO did not return any
    row.');
END;
```

7. Triggers

- What is a Trigger?

  A trigger is a pl/sql block structure which is fired when a DML statements like Insert, Delete, Update is executed on a database table. A trigger is triggered automatically when an associated DML statement is executed.

7. Triggers(cont)

- How to create a trigger?

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
BEGIN
  --- sql statements
END;
```

7. Triggers(cont)

- Example

The price of a product changes constantly. It is important to maintain the history of the prices of the products.

We can create a trigger to update the 'product_price_history' table when the price of the product is updated in the 'product' table.

1. Create the 'product' table and 'product_price_history' table

```
CREATE TABLE product_price_history
(product_id      NUMBER(5)
,product_name    VARCHAR2(32)
,supplier_name   VARCHAR2(32)
,unit_price      NUMBER(7,2) );
```

```
CREATE TABLE product
(product_id      NUMBER(5)
,product_name    VARCHAR2(32)
,supplier_name   VARCHAR2(32)
,unit_price      NUMBER(7,2) );
```

7. Triggers(cont)

- Example(cont)

2. Create the price_history_trigger and execute it.

```
CREATE OR REPLACE TRIGGER price_history_trigger
BEFORE UPDATE OF unit_price
ON product
FOR EACH ROW
BEGIN
     INSERT INTO product_price_history
     VALUES
     (:old.product_id,
      :old.product_name,
      :old.supplier_name,
      :old.unit_price);
END;
```

7. Triggers(cont)

- Example(cont)

3. Lets update the price of a product.

```
-- Once executed, the trigger fires and updates the
-- product_price_history' table
UPDATE PRODUCT SET unit_price = 800 WHERE product_id = 100
```

4. If you ROLLBACK the transaction before committing to the database, the data inserted to the table is also rolled back.

7. Triggers(cont)

- Types of PL/SQL Triggers
  - **Row level trigger** - An event is triggered for each row updated, inserted or deleted
  - **Statement level trigger** - An event is triggered for each SQL statement executed

- PL/SQL Trigger Execution Hierarchy
  - BEFORE statement trigger fires first
  - Next BEFORE row level trigger fires, once for each row affected
  - Then AFTER row level trigger fires once for each affected row. This events will alternates between BEFORE and AFTER row level triggers
  - Finally the AFTER statement level trigger fires

7. Triggers(cont)

- How To know Information about Triggers.
  - We can use the data dictionary view 'USER_TRIGGERS' to obtain information about any trigger.

```
DESC USER_TRIGGERS;
```

  - This view stores information about header and body of the trigger

```
SELECT * FROM user_triggers WHERE trigger_name =
 'trigger_name';
```

  - You can drop a trigger using the following command.

```
DROP TRIGGER trigger_name;
```

IV. Conclusion

Now, I hope this slide gave you some ideas about working with PL/SQL. Please let me know if there's any mistake or issue in the tutorial. All comments are welcome.

With our position in CTI Vietnam. This amount of knowledge would not be enough for us. There're many more topics we need to invest our time: performance tuning, data mining, coding convention, best practices, …
So, let's share our knowledge, TOGETHER.

V. Reference

1. PL/SQL Tutorial from plsql-tutorial.com
2. PL/SQL User's Guide and Reference, Release 2 (9.2), from Oracle
3. Database Interaction with PL/SQL, Jagadish Chatarji (Dev Shed)
4. PL/SQL Code Examples, ACS department, University of Utah
5. Doing SQL from PL/SQL: Best and Worst Practices, Oracle

VI. Keywords

Below is some keywords not mentioned in this slide. If possible, you should google them.

General: *Embedded SQL, Native dynamic SQL, The DBMS_Sql API, Name resolution, PL/SQL compiler, bounded and unbounded, oracle hint, materialized view.*

Cursors: *sharable SQL structure, session cursor, ref cursor, cursor variable, strong ref cursor, weak ref cursor, identified cursor, DBMS_Sql numeric cursor.*

Exception Handling: *RAISE_APPLICATION_ERROR, PRAGMA, EXCEPTION_INIT.*

Triggers: *CYCLIC CASCADING, ROLLBACK, INSTEAD OF, REFERENCING OLD.*

VII. Q&A

Questions, please

...