Lab 7

Procedural Language
Structured Query Language
(PL/SQL)

Eng. Mohammed Alokshiya

November 23, 2014
Structured Query Language (SQL) is the primary language used to access and modify data in relational databases. There are only a few SQL commands you can easily learn and use them. However, if you want to alter any data that is retrieved in a conditional manner, you soon encounter the limitations of SQL.

PL/SQL is designed to meet more requirements than SQL. It provides a programming extension to already-existing SQL.

PL/SQL defines a block structure for writing code. Maintaining and debugging the code is made easier with such a structure. One can easily understand the flow and execution of the program unit. PL/SQL offers modern software engineering features such as data encapsulation, exception handling, information hiding, and object orientation. It brings state-of-the-art programming to the Oracle server and toolset. PL/SQL provides all the procedural constructs that are available in any third-generation language (3GL).

**PL/SQL Block Structure**

A PL/SQL block consists of three sections:

- **Declarative (optional):** The declarative section begins with the keyword DECLARE and ends when the executable section starts.

- **Executable (required):** The executable section begins with the keyword BEGIN and ends with END. Observe that END is terminated with a semicolon. The executable section of a PL/SQL block can in turn include any number of PL/SQL blocks.

- **Exception handling (optional):** The exception section is nested within the executable section. This section begins with the keyword EXCEPTION.

<table>
<thead>
<tr>
<th>PL/SQL Block Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>DECLARE (optional)</td>
</tr>
<tr>
<td>-- Variables, cursors, user-defined exceptions</td>
</tr>
<tr>
<td>BEGIN (mandatory)</td>
</tr>
<tr>
<td>-- SQL statements</td>
</tr>
<tr>
<td>-- PL/SQL statements</td>
</tr>
<tr>
<td>EXCEPTION (optional)</td>
</tr>
<tr>
<td>--Actions to perform when errors occur -- like try-catch blocks in java</td>
</tr>
<tr>
<td>END; (mandatory)</td>
</tr>
</tbody>
</table>
Block Types

A PL/SQL program comprises one or more blocks. These blocks can be entirely separate or nested within another block. There are three types of blocks that make up a PL/SQL program. They are:

- Anonymous blocks
- Procedures
- Functions

### Anonymous blocks

Anonymous blocks are unnamed blocks. They are declared inline at the point in an application where they are to be executed and are compiled each time the application is executed. These blocks are not stored in the database. They are passed to the PL/SQL engine for execution at run time.

The other blocks, procedure and functions blocks, will be explained in the next lecture.

Example: create anonymous block to print “Hello World!” on the console

```sql
SET SERVEROUTPUT ON;
DECLARE
BEGIN
  DBMS_OUTPUT.PUT_LINE('Hello World!');
END;
```

The first line (`SET SERVEROUTPUT ON;`) is used to enable output in SQL Developer. We execute it one time only after logging to our database account.
To execute PL/SQL block, mark it and press the run button:

```sql
DECLARE
    EMP_HIREDATE DATE;
    EMP_DEPTNO NUMBER(2) NOT NULL := 10;
    LOCATION VARCHAR2(13) := 'Atlanta';
    C_COMM CONSTANT NUMBER := 1400;
BEGIN
    ...
END;
```

### Declaring PL/SQL Variables

You can declare variables in the declarative part of any PL/SQL block, subprogram, or package. Declarations allocate storage space for a value, specify its data type, and name the storage location so that you can reference it.

In the executable section, the existing value of the variable can be replaced with the new value.

#### Declaring & Initializing Variables Syntax

```plaintext
IDENTIFIER [CONSTANT] DATATYPE [NOT NULL] ::= [ DEFAULT EXPR];
```

#### Examples

```sql
DECLARE
    EMP_HIREDATE DATE;
    EMP_DEPTNO NUMBER(2) NOT NULL := 10;
    LOCATION VARCHAR2(13) := 'Atlanta';
    C_COMM CONSTANT NUMBER := 1400;
BEGIN
    ...
END;
```
Using Variables

```
DECLARE
  MYNAME VARCHAR2(8);
BEGIN
  DBMS_OUTPUT.PUT_LINE('My name is: ' || MYNAME);
  MYNAME := 'Mohammed';
  DBMS_OUTPUT.PUT_LINE('My name is: ' || MYNAME);
END;
```

Output

```
anonymous block completed
My name is:
My name is: Mohammed
```

The “SELECT INTO” Clause

The SELECT INTO clause is used to retrieves data from one or more database tables, and assigns the selected values to variables or collections. In its default usage (SELECT ... INTO), this statement retrieves one or more columns from only one row.

```
DECLARE
  LNAME VARCHAR2(20);
  SAL NUMBER(6);
BEGIN
  SELECT LAST_NAME, SALARY
  INTO LNAME, SAL
  FROM EMPLOYEES
  WHERE EMPLOYEE_ID = 110;
  DBMS_OUTPUT.PUT_LINE(LNAME || ' ' || SAL);
END;
```

Output

```
anonymous block completed
Chen 8200
```

The %TYPE Attribute

The %TYPE attribute is used to declare a variable according to:

- A database column definition
- Another declared variable

It is a prefixed with:
- The database table and column
- The name of the declared variable

Use it to declare a new variable with the same data type of a predefined variable or a column in a table.

### Syntax

```
IDENTIFIER TABLE_NAME.COLUMN_NAME%TYPE;
```

### Examples

```
DECLARE
  EMP_LNAME EMPLOYEES.LAST_NAME%TYPE;
  BALANCE NUMBER(7,2);
  MIN_BALANCE BALANCE%TYPE := 1000;
BEGIN
  ...
END;
```

### Control Structures

#### IF Statements

### Syntax

```
IF CONDITION THEN
  STATEMENTS;
[ELSIF CONDITION THEN
  STATEMENTS;] 
[ELSE
  STATEMENTS;]
END IF;
```

### Example

```
DECLARE
  GRADE NUMBER(3) := 95;
  RESULT VARCHAR2(1);
BEGIN
  IF GRADE >= 90 THEN
    RESULT := 'A';
  ELSIF GRADE >= 80 THEN
    RESULT := 'B';
  ELSIF GRADE >= 70 THEN
    RESULT := 'C';
  ELSIF GRADE >= 60 THEN
    RESULT := 'D';
  ELSE
    RESULT := 'F';
  END IF;
  DBMS_OUTPUT.PUT_LINE('Result: ' || RESULT);
END;
```
**CASE Expression**

A CASE expression selects a result and returns it. To select the result, the CASE expression uses expressions. The value returned by these expressions is used to select one of several alternatives.

### Syntax

```sql
CASE SELECTOR
  WHEN EXPRESSION1 THEN RESULT1
  WHEN EXPRESSION2 THEN RESULT2
  ...
  WHEN EXPRESSIONN THEN RESULTN
  [ELSE RESULTN+1]
END;
```

### Example

```sql
DECLARE
  GRADE CHAR(1) := 'A';
  APPRAISAL VARCHAR2(20);
BEGIN
  APPRAISAL :=
    CASE GRADE
      WHEN 'A' THEN 'Excellent'
      WHEN 'B' THEN 'Very Good'
      WHEN 'C' THEN 'Good'
      ELSE 'No such grade'
    END;
  DBMS_OUTPUT.PUT_LINE ('Grade: ' || GRADE || ' - Appraisal ' || APPRAISAL);
END;
```

**Searched CASE Expressions**

In searched CASE statements, you do not have a test expression. Instead, the WHEN clause contains an expression that results in a Boolean value. The same example is rewritten in this slide to show searched CASE statements.

### Example

```sql
DECLARE
  GRADE CHAR(1) := 'B';
  APPRAISAL VARCHAR2(20);
BEGIN
  APPRAISAL :=
    CASE
      WHEN GRADE = 'A' THEN 'Excellent'
      WHEN GRADE IN ('B', 'C') THEN 'Good'
      ELSE 'No such grade'
    END;
  DBMS_OUTPUT.PUT_LINE ('Grade: ' || GRADE || ' - Appraisal ' || APPRAISAL);
END;
```
Logic Tables

<table>
<thead>
<tr>
<th>AND</th>
<th>TRUE</th>
<th>FALSE</th>
<th>NULL</th>
<th>OR</th>
<th>TRUE</th>
<th>FALSE</th>
<th>NULL</th>
<th>NOT</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>NULL</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>NULL</td>
<td>FALSE</td>
</tr>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>FALSE</td>
<td>NULL</td>
<td>NULL</td>
<td>TRUE</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>

Basic Loops

Syntax

```sql
LOOP
  STATEMENT1;
  ...
  EXIT [WHEN CONDITION];
END LOOP;
```

Example: write a PL/SQL code to print number from 1 to 15 to the console.

```sql
DECLARE
  I NUMBER(2) := 1;
BEGIN
  LOOP
    DBMS_OUTPUT.PUT_LINE (I);
    I := I + 1;
    EXIT WHEN I > 15;
  END LOOP;
END;
```
**WHILE Loops**

**Syntax**

```
WHILE CONDITION LOOP
    STATEMENT1;
    STATEMENT2;
    ...
END LOOP;
```

Example: write a PL/SQL code to print number from 1 to 15 to the console.

```
DECLARE
    I NUMBER(2) := 1;
BEGIN
    WHILE I <= 15 LOOP
        DBMS_OUTPUT.PUT_LINE (I);
        I := I + 1;
    END LOOP;
END;
```

**FOR Loops**

You can use a FOR loop to shortcut the test for the number of iterations. You do not have to declare the counter; it is declared implicitly. 'lower_bound .. upper_bound' is required syntax.

**Syntax**

```
FOR COUNTER IN [REVERSE] LOWER_BOUND..UPPER_BOUND LOOP
    STATEMENT1;
    STATEMENT2;
    ...
END LOOP;
```

Example: write a PL/SQL code to print number from 1 to 15 to the console.

```
DECLARE
BEGIN
    FOR I IN 1..15 LOOP
        DBMS_OUTPUT.PUT_LINE (I);
    END LOOP;
END;
```