Chapter 5:Advanced SQL

Dr. Abu Raihan Mostofa Kamal

August 30, 2021

PL/SQL Block

Block: Smallest unit of code

In PL/SQL, as in most other procedural languages, **the smallest meaningful grouping of code** is known as a block. A block is a unit of code that provides **execution and scoping boundaries for variable declarations and exception handling**.

Blocks: Anonymous

It has no Name

It cannot be called by any other block—it doesn't have a handle for reference. Instead, anonymous blocks serve as containers that execute PL/SQL statements. It is **not stored** in the database and **can not be reused**. (one-time execution).

```
BEGIN

DBMS_OUTPUT.put_line ('Hello World!');

END;
```

Block: Named

It has Name

While anonymous PL/SQL blocks are indispensable, the majority of code you write will be in named blocks. **Examples:** Functions and Procedure. It is stored in the database and can be reused.

Conditional Statements

If then is the most primitive of conditional statements.

Syntax:

```
  IF condition
THEN
... TRUE sequence of executable statements ...
ELSE
... FALSE/NULL sequence of executable statements ...
END IF;
```

Conditional Statement

If then else is the extension of the basic if then. Remember the spelling of **ELSIF**.

```
  IF condition-1
THEN
statements-1
ELSIF condition-N
THEN
statements-N
[ELSE
else_statements]
END IF;
```

Function

Modular Code using Function

- More reusable
- More manageable
- More readable

Functions (Cont.)

Structure of a Function:

```
  FUNCTION [schema.]name[( parameter[, parameter...] ) ]
RETURN return_datatype
IS
[declaration statements]

BEGIN
executable statements
[EXCEPTION
exception handler statements]

END [name];
```

A few points about Functions

- IN parameter (like conventional parameter), OUT parameter (not used here. used for writing the result back to the caller)
- No datatype precision is allowed in parameter and return type (**...return varchar2(20) is not valid**)
- Function Name and parameters should be **self-explanatory**. (**do not use abc as function name, x, y , z as parameters**)

Function: A simple example

```
 CREATE OR REPLACE FUNCTION get_total_sales
RETURN NUMBER
IS
    v_total_sales NUMBER := 0;
BEGIN
    -- get total sales
    SELECT SUM(unit_price * quantity)
    INTO l_total_sales
    FROM order_items
    INNER JOIN orders USING (order_id)
    WHERE status = 'Shipped';


    -- return the total sales
    RETURN v_total_sales;
END;
```

Calling a function

There are 3 general options:

1. It can be readily tested by a fixed parameter from dual:
   ```
   select get_total('CSE') from dual;
   ```
2. You can test it by passing it as a parameter in put_line function:
   ```
   DBMS_OUTPUT.PUT_LINE('Sales: ' || get_total_sales);
   ```
3. It can be selected from a table values, the parameter values can be passed dynamically in the select statement.
   ```
   select id, get_cgpa(sid) from students where dept='CSE';
   ```

DECODE function

```
DECODE(expr, search, result
          [, search, result ]...
     [, default ]
    )
```

DECODE()

DECODE compares expr to each search value one by one. If expr is equal to a search, then Oracle Database returns the corresponding result. If no match is found, then Oracle returns default. If default is omitted, then Oracle returns null.

DECODE by example

```
SELECT product_id,
      DECODE (warehouse_id, 1, 'Southlake',
                            2, 'San Francisco',
                            3, 'New Jersey',
                            4, 'Seattle',
                               'Non domestic')
      "Location of inventory" FROM inventories
      WHERE product_id < 100;
```

RANK() function

- The **RANK()** function is **an analytic function** that calculates the rank of a value in a set of values.
- The RANK() function returns the same rank for the rows with the same values. It adds the number of tied rows to the tied rank to calculate the next rank. Therefore, the ranks **may not be consecutive numbers**.
- The RANK() function is useful for **top-N** and **bottom-N** queries.

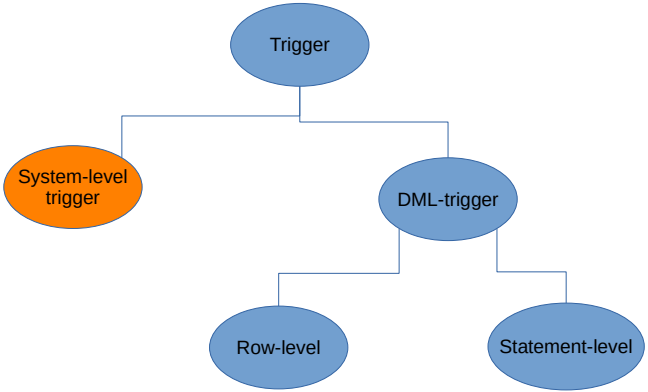
RANK() Syntax and Example:

```
RANK()
        OVER ([ query_partition_clause ] order_by_clause)

Example:

select sid,name,cgpa,
rank() over (order by cgpa desc) position
from students;
```

Triggers (Elementary)

- A trigger is a statement that is executed automatically by the system as a side effect of a modification to the database.
- To design a trigger mechanism, we must:
  - Specify the **conditions** under which the trigger is to be executed.
  - Specify the **actions** to be taken when the trigger executes.
- Since trigger is event-oriented, it is **not explicitly called** like functions rather it **automatically fires** wherever the **condition is met**.

Trigger : Broad Classification



Row-level Trigger

Some examples:

- Whenever an account holder withdraws money, his balance must be greater than the withdrawn amount.
- Any student is admitted with basic information, his/her Student ID will be generated automatically.
- Whenever any update (1 or more rows affected) occurs in a specific table , the time of update should be recorded without human intervention.

So, they have one in common: **it is executed for each possible record's change.**

Row-level Trigger (Cont.), Syntax

Indicator: for each row clause

```
CREATE OR REPLACE TRIGGER trigger_name
  BEFORE | AFTER
  INSERT OR DELETE OR UPDATE OF column1, column2,
  ON table_name
  FOR EACH ROW
  REFERENCING OLD AS old_name
  NEW AS new_name
  WHEN (condition)
  DECLARE

BEGIN

EXCEPTION

END;
```

Statement-level Trigger

It works exactly in the same way as except it is executed only once no matter how many records are affected by the DML statement.

Syntax: Just omit the **for each row** option.