Marwadi University

01CT0407 - Database Management System

PL / SQL

Prof. Harikesh Chauhan
Information Communication and Technology

Outline

- View
- PL/SQL blocks,
- PL/SQL data types,
- Conditional statements and looping,
- SQL within PL/SQL,
- Error Handling Cursors,
- Stored Procedures and Stored Function,
- Database Triggers

Views

- The view is a virtual table. It does not physically exist. Rather, it is created by a query joining one or more tables.
- A view contains rows and columns, just like a real table
- The fields in a view are fields from one or more real tables in the database

Creating an SQL VIEW

Syntax:

```
CREATE VIEW view_name AS
SELECT column_name(s)
FROM table_name
WHERE condition;
```

Views

View Creation – Example

1. CREATE VIEW transaction_view
   AS SELECT acc_no, tr_date,amt from transaction;

2. Create view account_view
   AS SELECT a.acc_no,a.balance,l.loan_amt,l.interest_rate from
   account a, loan l where a.acc_no=l.acc_no and a.city='Ahmedabad';

The above views would create a virtual table based on the result set of the select statement. To show created views:

1. SELECT * FROM transaction_view;
2. SELECT * FROM account_view;

Views

Updating View

You can modify the definition of a VIEW without dropping it by using the following syntax:

```
CREATE OR REPLACE VIEW view_name
AS SELECT columns
FROM table
WHERE predicates;
```

1. Create or Replace view account_view
   AS SELECT
   a.acc_no,a.balance,l.loan_amt,l.interest_rate,
   l.remaining_loan from account a, loan l where
   a.acc_no=l.acc_no and **a.city='Mehsana'**;

Views

Deleting View

The syntax for dropping a VIEW :

DROP VIEW view_name;

View Drop - Example

DROP VIEW transaction_view;

Views

Sr. No.	Key	Views	Materialized Views
1	Definition	Technically View of a table is a logical virtual copy of the table created by "select query" but the result is not stored anywhere in the disk and every time we need to fire the query when we need data, so always we get updated or latest data from original tables.	On other hand Materialized views are also the logical virtual copy of data-driven by the select query but the result of the query will get stored in the table or disk.
2	Storage	In Views the resulting tuples of the query expression is not get storing on the disk only the query expression is stored on the disk.	On other hand in case of Materialized views both query expression and resulting tuples of the query get stored on the disk.
3	Query Execution	As mentioned above in case of Views the query expression is stored on the disk and not its result so query expression get executed every time when user try to fetch data from it so that user will get the latest updated value every time.	While on other hand in case of Materialized Views the result of query is get stored on the disk and hence the query expression did not get executed every time when user try to fetch the data so that user will not get the latest updated value if it get changed in database.
4	Cost Effective	As Views does not have any storage cost associated with it so they also does not have any update cost associated with it.	On other hand Materialized Views does have a storage cost associated with it so also have update cost associated with it.
5	Design	Views in SQL are designed with a fixed architecture approach due to which there is an SQL standard of defining a view.	On other hand in case of Materialized Views in SQL are designed with a generic architecture approach so there is no SQL standard for defining it,and its functionality is provided by some databases systems as an extension.
6	Usage	Views are generally used when data is to be accessed infrequently and data in table get updated on frequent basis.	On other hand Materialized Views are used when data is to be accessed frequently and data in table not get updated on frequent basis.

PL / SQL

- PL/SQL stands for "**Procedural Language extensions to the Structured Query Language**."
- SQL is a popular language for both querying and updating data in relational database management systems (RDBMS).
- PL/SQL is a highly structured and readable language.
- PL/SQL is a standard and portable language for Oracle Database development.
- PL/SQL program that runs on a system that does not have an Oracle Database.
- PL/SQL is a high-performance and highly integrated database language.

The PL/SQL Comments

- **Single-line comments**

Syntax:

-- valued added tax 10%

- **Multi-line comments**

Syntax:

```
/*
   PL/SQL executable statement(s)
  */
```

PL/SQL BLOCK STRUCTURE

DECLARE (optional)

- variable declarations

BEGIN (required)

- SQL statements
- PL/SQL statements or sub-blocks

EXCEPTION (optional)

- actions to perform when errors occur

END; (required)

PL/SQL

- There are 2 types of Datatypes:

1. Scalar (Char, Varchar2, Date and Number)
2. Composite (%rowtype)

- All Variables required to be used in the program must be declared before their use. Variable names cannot be table, column and keyword names.

- Assignment Operator is denoted as ":=" and each statement of PL/SQL ends with ";".

- Dbms_Output.Put_Line() is used for printing Output.

- &Variable_Name is used to take input from the user.

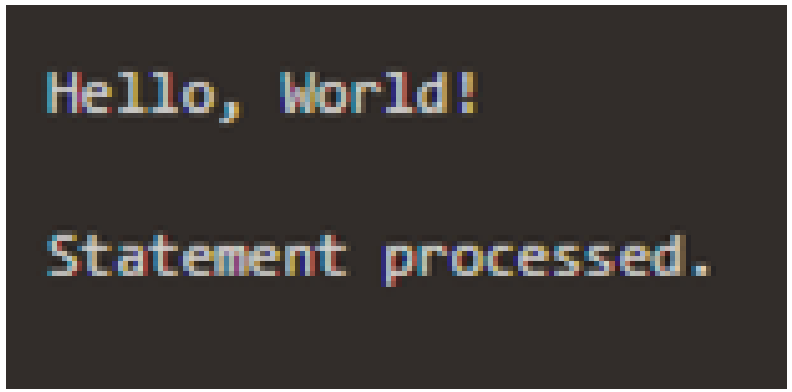
PL / SQL Syntax

- Syntax:

```
DECLARE
  <declarations section>
BEGIN
  <executable command(s)>
EXCEPTION
  <exception handling>
END;
```

- **Example: Print The 'Hello World'**

```
DECLARE
   message  varchar2(20):= 'Hello, World!';
BEGIN
   dbms_output.put_line(message);
END;
```



PL / SQL Example

- **Example: Calculator**

```
-- basic Program
DECLARE
   a integer := 30;
   b integer := 40;
   c integer;
   f real;
BEGIN
   c := a + b;
   dbms_output.put_line('Value of c: ' || c);
   f := 100.0/3.0;
   dbms_output.put_line('Value of f: ' || f);
END;
/
```

```
Value of c: 70
Value of f: 33.333333333333333333333333333333333333

Statement processed.
```

The PL/SQL IF Statement

- **PL/SQL IF THEN statement**

```
IF condition THEN
   statements;
END IF;
```

Example:

```
DECLARE
            x NUMBER := 20;
BEGIN
  IF x > 10 THEN
    DBMS_OUTPUT.PUT_LINE( 'Sales revenue is greater than 10 ' );
  END IF;
END;
/
```

```
Sales revenue is greater than 10

Statement processed.
```

The PL/SQL IF Statement

Marwadi University

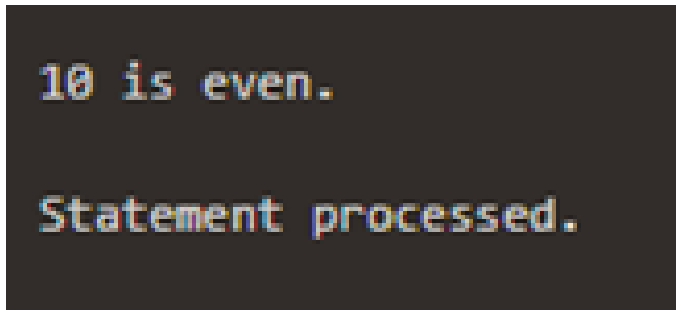
- **PL/SQL IF THEN ELSE statement**

```
IF condition THEN
   statements;
ELSE
   elsestatements;
END IF;
```

Example:

```
DECLARE
  v_number NUMBER := 10; -- Change this to the number you want to check

BEGIN
  -- Check if the number is even
  IF MOD(v_number, 2) = 0 THEN
    DBMS_OUTPUT.PUT_LINE(v_number || ' is even.');
  ELSE
    DBMS_OUTPUT.PUT_LINE(v_number || ' is odd.');
  END IF;
END;
/
```



The PL/SQL IF Statement

- **PL/SQL Nested IF statement**

```
IF condition1 THEN
   IF condition2 THEN
      nestedifstatements;
   END IF;
ELSE
   elsestatements;
END IF;
```

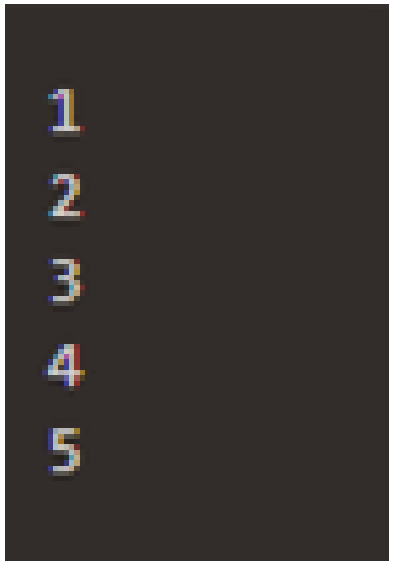
The PL/SQL Case

PL/SQL CASE…

```
CASE
WHEN VARIABLE = VALUE THEN
   <Code>
WHEN VARIABLE = VALUE THEN
   <Code>
ELSE
    <Code>
END CASE;
```

The PL/SQL FOR LOOP

```
FOR  <Variable> IN <Min> .. <Max>
Loop
<Code>
End Loop;
```



Example: Print 1 to 5

```
BEGIN
  FOR l_counter IN 1..5
  LOOP
    DBMS_OUTPUT.PUT_LINE( l_counter );
  END LOOP;
END;
```

The PL/SQL WHILE LOOP

```
WHILE condition
LOOP
   statements;
END LOOP;
```

```
Counter : 1
Counter : 2
Counter : 3
Counter : 4
Counter : 5
```

Example: Print 1 to 5

```
DECLARE
  n_counter NUMBER := 1;
BEGIN
  WHILE n_counter <= 5
  LOOP
    DBMS_OUTPUT.PUT_LINE( 'Counter : ' || n_counter );
    n_counter := n_counter + 1;
  END LOOP;
END;
/
```

PL/SQL Examples

Write a PL/SQL block to find maximum of 2 numbers

```
declare
x number(5);
y number(5);
begin
  x:=10;
  y:=20;
if x > y then
    DBMS_OUTPUT.PUT_LINE(''|| x ||' is greater than '|| y ||'');
else
    DBMS_OUTPUT.PUT_LINE(''|| y ||' is greater than '|| x ||'');
end if;
end;
```

PL/SQL Examples

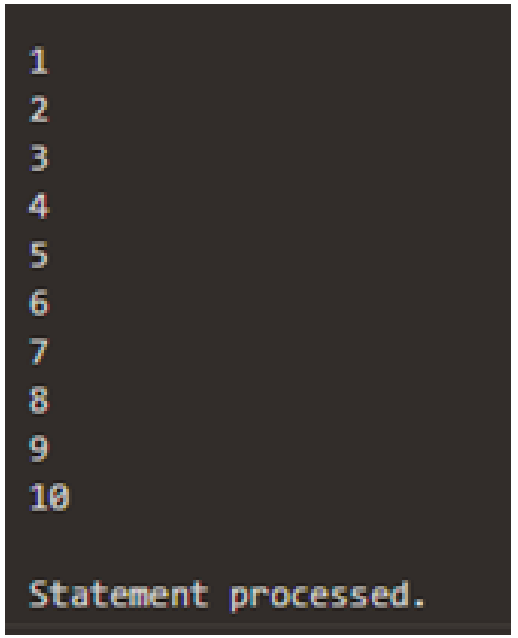
Write a PL/SQL block to find area of rectangle, square and circle

```
declare
  area_c   number(6, 2)  ;
  radius number(1) := 3 ;
 area_r number(6) ;
area_s number(6) ;
length number(2) ;
width number(2) ;
  pi constant number(3, 2) := 3.14;
  begin
        area_c := pi * radius * radius;
        DBMS_OUTPUT.PUT_LINE('Circle area = ' || area_c);
        area_r := length * width;
        DBMS_OUTPUT.PUT_LINE('Rectangle area = ' || area_r);
         area_s := length * length;
        DBMS_OUTPUT.PUT_LINE('Square area = ' || area_s);
 end;
```

The PL/SQL EXAMPLE

Example: Write a Pl/SQL program to print integers from 1 to 10 by using PL/SQL FOR loop

```
DECLARE
  n number:= 10;
BEGIN
     for i in 1..n loop
       DBMS_OUTPUT.PUT_LINE(i);
    END LOOP;
 END;
/
```

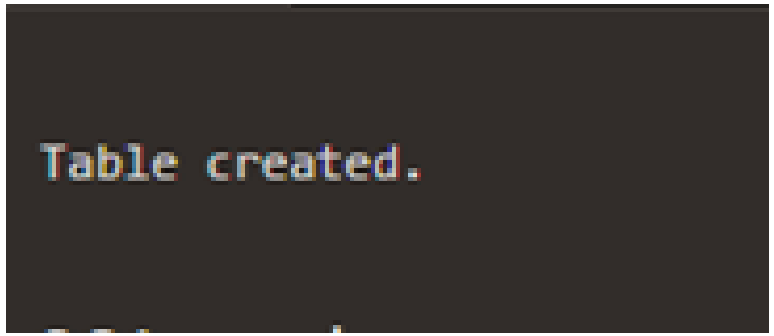


The PL/SQL EXAMPLE

Example: Write a Pl/SQL program using FOR loop to insert ten rows into a database table.

Step 1: Create a temp table as follow:

```
create table temp1
(
    no number(5,2),
    detail varchar2(50)
);
```



The PL/SQL EXAMPLE

Step 2: Write PL/SQL for Inserting ten Rows

```
DECLARE
BEGIN
   FOR i IN 1..10 LOOP
      IF MOD(i,2) = 0 THEN     -- i is even
         INSERT INTO temp1 VALUES (i, 'i is even');
      ELSE
         INSERT INTO temp1 VALUES (i, 'i is odd');
      END IF;
   END LOOP;
   COMMIT;
END;
/
```

```
1 row(s) inserted.
```

The PL/SQL EXAMPLE

Step 3: To check the data is inserted into table:

select * from temp1;

NO	DETAIL
1	i is odd
2	i is even
3	i is odd
4	i is even
5	i is odd
6	i is even
7	i is odd
8	i is even
9	i is odd
10	i is even

PL/SQL Sample Program (with user input)

```
DECLARE
  v_inv_value  number(8,2);
 v_price      number(8,2);
  v_quantity   number(8,0)  :=  400;
BEGIN
   v_price := &V_price;
  v_inv_value := v_price * v_quantity;
   dbms_output.put_line('******');
   dbms_output.put_line('price * quantity=');
   dbms_output.put_line(v_inv_value);
END;
/
```

Note: PL/SQL not designed for user interface programming

PL/SQL

%RowType

Variable Name TableName%RowType

acc account%RowType;

- This “acc” variable will have complete structure of the table account. Any column of account table can be referred using “acc” variable

```
DECLARE
 acc  account%RowType;
BEGIN
 select * into acc  from account where acc_no=‘A001’;
 DBMS_OUTPUT.PUT_LINE (acc.name);
 DBMS_OUTPUT.PUT_LINE (acc.city);
 DBMS_OUTPUT.PUT_LINE (acc.balance);
 DBMS_OUTPUT.PUT_LINE (acc.loan_taken);
END;
```

Stored Procedure(Stored Programs)

- A program unit is a self-contained group of program statements that can be used within a large program
- Stored PL/SQL program units are program units that other PL/SQL program can reference and that other database users can use and execute.
- Can receive multiple input parameters and return multiple output values (or no output values) and perform DML Commands

```
CREATE [OR REPLACE] PROCEDURE procedure_name [
(parameter mod datatype [,parameter]) ]
   IS /AS
        [local variable declaration_section]
  BEGIN
        executable_section
  [EXCEPTION
        exception_section]
  END [procedure_name];
```

Header Section

body

Exception section

Stored Procedure(Stored Programs)

- The header section defines:
  1. The procedure name (name should be unique)
  2. The parameters that the procedure receives or delivers (IN, OUT, IN OUT)
  3. IS keyword: follows the parameters list
  4. The local procedure variables
- Create or replace: instructs the DBMS to create the procedure if it does it exists, otherwise; it replaces the existing one
- OR REPLACE clause is optional. But if omitted and a procedure exists with the same name, an error occurs
- Parameter name, mod and data type are enclosed in parentheses, each is separated by a comma.
- Parameter mod: describes how the procedure change the value. It can be:
  1. IN: passed parameter is a Read only value. Cannot be changed by the procedure
  2. OUT: write-only value. Always comes on the left side of an assignment statement
  3. IN OUT: its value can be changed

Stored Procedure(Stored Programs)

EXAMPLE:

```
CREATE OR REPLACE PROCEDURE
greetings
AS
BEGIN
  dbms_output.put_line('Hello World!');
END;
```

The following example creates a simple procedure that displays the string 'Hello World!' on the screen when executed.

EXECUTE greetings;

Command to Execute the Store Procedure

Output: Hello World!

Stored Procedure(Stored Programs)

EXAMPLE:

```
Create or replace procedure account_update
(
   curr_acc_no IN varchar2,
   curr_name IN varchar2,
   curr_balance IN number
)
IS
BEGIN
          update account
          set balance = curr_balance
          where
          acc_no = curr_acc_no and
          name = curr_name;
          COMMIT;
END;
```

The Following Stored Procedure updates the balance of the account for a given account number and name of the account.

Command to Execute the Store Procedure

Execute account_update('A001','Patel',125000);

The PL/SQL Procedure Example

- To delete Procedure:
- Syntax: DROP PROCEDURE procedure-name;
- Example: DROP PROCEDURE greetings;

The PL/SQL Procedure Example

- **Example: Creates a simple procedure that displays the string 'Hello World!' on the screen when executed.**

```
CREATE OR REPLACE PROCEDURE greetings
AS
BEGIN
   dbms_output.put_line('Hello World!');
END;
/
```

```
Procedure created.
```

The PL/SQL Procedure Example

To execute Procedure:

Syntax:

```
BEGIN
   Procedure_Name;
END;
/
```

or

```
EXECUTE Procedure_Name;
```

```
Hello World!

Statement processed.
```

Example:

```
BEGIN
```

The PL/SQL Procedure Example

- **Write procedure computes the square of value of a passed value. (This example shows how we can use the same parameter to accept a value and then return another result.)**

```
DECLARE
   a number;
PROCEDURE squareNum(x IN OUT number) IS
BEGIN
  x := x * x;
END;
BEGIN
   a:= 23;
   squareNum(a);
   dbms_output.put_line(' Square of (23): ' || a);
END;
/
```

```
 Square of (23): 529

Statement processed.
```

The PL/SQL Procedure Example

- **Write a Procedure that finds the minimum of two values. (Here, the procedure takes two numbers using the IN mode and returns their minimum using the OUT parameters.)**

```
DECLARE
   a number;
   b number;
   c number;
PROCEDURE findMin(x IN number, y IN number, z OUT number) IS
BEGIN
   IF x < y THEN
      z:= x;
   ELSE
      z:= y;
   END IF;
END;
BEGIN
   a:= 23;
   b:= 45;
   findMin(a, b, c);
   dbms_output.put_line(' Minimum of (23, 45) : ' || c);
END;
```

```
 Minimum of (23, 45) : 23

Statement processed.
```

The PL/SQL Procedure Example

- Create the procedure for palindrome of given number

```
CREATE OR REPLACE PROCEDURE pali(n IN NUMBER) IS
   m NUMBER;
   temp NUMBER := 0;
   rem NUMBER;
   n_copy NUMBER := n; -- Declare a copy of the input parameter
BEGIN
   m := n_copy;
   -- while loop with condition till n_copy > 0
   WHILE n_copy > 0
   LOOP
      rem := MOD(n_copy, 10);
      temp := (temp * 10) + rem;
      n_copy := TRUNC(n_copy / 10);
   END LOOP; -- end of while loop here

   IF m = temp THEN
      dbms_output.put_line('true');
```

```
Procedure created.
```

```
true

Statement processed.
```

Cursor

- A cursor is a private set of records.
- A cursor is a pointer that points to a result of a query.
- A cursor is a temporary work area created in the system memory when a SQL statement is executed. A cursor contains information on a select statement and the rows of data accessed by it.
- This temporary work area is used to store the data retrieved from the database, and manipulate this data. A cursor can hold more than one row, but can process only one row at a time. The set of rows the cursor holds is called the *active* set.
- There are 2 types of Cursors:

1. Implicit Cursor: Created automatically for every query in SQL.
2. Explicit Cursor: Created manually by the programmer.

Cursor

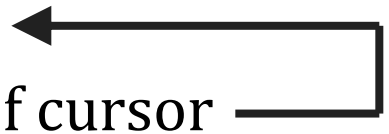
Cursor Attributes

cursorname%ROWCOUNT	Rows returned so far
cursorname%FOUND	One or more rows retrieved
cursorname%NOTFOUND	No rows found
Cursorname%ISOPEN	Is the cursor open

Cursor

Explicit Cursor

- Declare the cursor
- Open the cursor
- Fetch a row
- Test for end of cursor
- Close the cursor



Cursor

Explicit Cursor Example

```
DECLARE
 CURSOR account_cursor IS SELECT * from account;
 acc account%rowtype;
BEGIN
 DBMS_OUTPUT.PUT_LINE ('******************');
 OPEN account_cursor;
 FETCH account_cursor into acc;
 WHILE account_cursor % Found
LOOP
  DBMS_OUTPUT.PUT_LINE (acc.name);
  DBMS_OUTPUT.PUT_LINE (acc.city);
  DBMS_OUTPUT.PUT_LINE (acc.balance);
  DBMS_OUTPUT.PUT_LINE (acc.loan_taken);
  DBMS_OUTPUT.PUT_LINE ('******************');
   FETCH account_cursor into acc;
END LOOP;
 CLOSE account_cursor;
END;
```

Create a Table SALESMEN as per Practical 8

Marwadi University

```
CREATE TABLE SALESMEN (
  SNUM VARCHAR2(6) CONSTRAINT pksnum PRIMARY KEY CHECK (SNUM LIKE 'S%'),
  SNAME VARCHAR2(20) NOT NULL,
  CITY VARCHAR2(15),
  COMM NUMBER(5, 2)
);
-- Insert data into the SALESMEN table
INSERT INTO SALESMEN (SNUM, SNAME, CITY, COMM) VALUES ('S1001', 'Piyush', 'London', 0.12);
INSERT INTO SALESMEN (SNUM, SNAME, CITY, COMM) VALUES ('S1002', 'Niraj', 'San Jose', 0.13);
INSERT INTO SALESMEN (SNUM, SNAME, CITY, COMM) VALUES ('S1003', 'Miti', 'London', 0.11);
INSERT INTO SALESMEN (SNUM, SNAME, CITY, COMM) VALUES ('S1004', 'Rajesh', 'Barcelona', 0.15);
INSERT INTO SALESMEN (SNUM, SNAME, CITY, COMM) VALUES ('S1005', 'Haresh', 'New York', 0.10);
INSERT INTO SALESMEN (SNUM, SNAME, CITY, COMM) VALUES ('S1006', 'Ram', 'Bombay', 0.10);
INSERT INTO SALESMEN (SNUM, SNAME, CITY, COMM) VALUES ('S1007', 'Nehal', 'Delhi', 0.09);


Select * from SALESMEN;
```

Create a Table CUSTOMER as per Practical 8


```
CREATE TABLE CUSTOMER (
  CNUM VARCHAR2(6) CONSTRAINT pkcnum PRIMARY KEY CHECK (CNUM LIKE 'C%'),
  CNAME VARCHAR2(20) NOT NULL,
  CITY VARCHAR2(15),
  RATING NUMBER(5),
  SNUM VARCHAR2(6) REFERENCES SALESMEN(SNUM)
);
-- Insert data into the CUSTOMER table
INSERT INTO CUSTOMER (CNUM, CNAME, CITY, RATING, SNUM) VALUES ('C2001', 'Hardik', 'London', 100, 'S1001');
INSERT INTO CUSTOMER (CNUM, CNAME, CITY, RATING, SNUM) VALUES ('C2002', 'Geeta', 'Rome', 200, 'S1003');
INSERT INTO CUSTOMER (CNUM, CNAME, CITY, RATING, SNUM) VALUES ('C2003', 'Kavish', 'San Jose', 200, 'S1002');
INSERT INTO CUSTOMER (CNUM, CNAME, CITY, RATING, SNUM) VALUES ('C2004', 'Dhruv', 'Berlin', 300, 'S1002');
INSERT INTO CUSTOMER (CNUM, CNAME, CITY, RATING, SNUM) VALUES ('C2005', 'Pratham', 'London', 100, 'S1001');
INSERT INTO CUSTOMER (CNUM, CNAME, CITY, RATING, SNUM) VALUES ('C2006', 'Vyomesh', 'San Jose', 300, 'S1007');
INSERT INTO CUSTOMER (CNUM, CNAME, CITY, RATING, SNUM) VALUES ('C2007', 'Kirit', 'Rome', 100, 'S1004');
INSERT INTO CUSTOMER (CNUM, CNAME, CITY, RATING, SNUM) VALUES ('C2008', 'Kirit', 'Rome', 100, 'S1004');
select * from Customer
```

Create a Table Order as per Practical 8

```
CREATE TABLE "ORDER" (
  ONUM VARCHAR2(6) CONSTRAINT pkonum PRIMARY KEY CHECK (ONUM LIKE 'O%'),
  AMT NUMBER(10, 2) NOT NULL,
  ODATE DATE,
  CNUM VARCHAR2(6) REFERENCES CUSTOMER(CNUM),
  SNUM VARCHAR2(6) REFERENCES SALESMEN(SNUM)
);
-- Insert data into the ORDER table
INSERT INTO "ORDER" (ONUM, AMT, ODATE, CNUM, SNUM) VALUES ('O3001', 18.69, TO_DATE('10-Mar-90', 'DD-Mon-RR'), 'C2008', 'S1007');
INSERT INTO "ORDER" (ONUM, AMT, ODATE, CNUM, SNUM) VALUES ('O3003', 767.19, TO_DATE('10-Mar-90', 'DD-Mon-RR'), 'C2001', 'S1001');
INSERT INTO "ORDER" (ONUM, AMT, ODATE, CNUM, SNUM) VALUES ('O3002', 1900.10, TO_DATE('03-Oct-90', 'DD-Mon-RR'), 'C2007', 'S1004');
INSERT INTO "ORDER" (ONUM, AMT, ODATE, CNUM, SNUM) VALUES ('O3005', 5160.45, TO_DATE('04-Oct-90', 'DD-Mon-RR'), 'C2003', 'S1002');
INSERT INTO "ORDER" (ONUM, AMT, ODATE, CNUM, SNUM) VALUES ('O3006', 1098.16, TO_DATE('10-Mar-90', 'DD-Mon-RR'), 'C2008', 'S1007');
INSERT INTO "ORDER" (ONUM, AMT, ODATE, CNUM, SNUM) VALUES ('O3009', 1713.23, TO_DATE('10-Apr-90', 'DD-Mon-RR'), 'C2002', 'S1003');
INSERT INTO "ORDER" (ONUM, AMT, ODATE, CNUM, SNUM) VALUES ('O3007', 75.75, TO_DATE('10-Apr-90', 'DD-Mon-RR'), 'C2004', 'S1002');
INSERT INTO "ORDER" (ONUM, AMT, ODATE, CNUM, SNUM) VALUES ('O3008', 4723.00, TO_DATE('10-May-90', 'DD-Mon-RR'), 'C2006', 'S1001');
INSERT INTO "ORDER" (ONUM, AMT, ODATE, CNUM, SNUM) VALUES ('O3010', 1309.95, TO_DATE('10-May-90', 'DD-Mon-RR'), 'C2004', 'S1002');
INSERT INTO "ORDER" (ONUM, AMT, ODATE, CNUM, SNUM) VALUES ('O3011', 9891.88, TO_DATE('10-Jun-90', 'DD-Mon-RR'), 'C2006', 'S1001');
Select * from "ORDER";
```

Given the table ORDER (ONUM, AMT, ODATE, CNUM, SNUM) write a cursor to select the five highest amount (AMT) order details from the table.

```
DECLARE
  CURSOR top_orders_cursor IS
    SELECT ONUM, AMT, ODATE, CNUM, SNUM
    FROM "ORDER"
    ORDER BY AMT DESC
    FETCH FIRST 5 ROWS ONLY;
  v_onum "ORDER".ONUM%TYPE;
  v_amt "ORDER".AMT%TYPE;
  v_odate "ORDER".ODATE%TYPE;
  v_cnum "ORDER".CNUM%TYPE;
  v_snum "ORDER".SNUM%TYPE;
BEGIN
  OPEN top_orders_cursor;
  FETCH top_orders_cursor INTO v_onum, v_amt, v_odate, v_cnum, v_snum;
  DBMS_OUTPUT.PUT_LINE('Top 5 Orders with the Highest Amounts:');
  DBMS_OUTPUT.PUT_LINE('ONUM' || CHR(9) || 'AMT' || CHR(9) || 'ODATE' || CHR(9) || 'CNUM' || CHR(9) || 'SNUM');
  DBMS_OUTPUT.PUT_LINE('-------------------------------------------');
    WHILE top_orders_cursor%FOUND LOOP
    DBMS_OUTPUT.PUT_LINE(v_onum || CHR(9) || v_amt || CHR(9) || v_odate || CHR(9) || v_cnum || CHR(9) || v_snum);
    FETCH top_orders_cursor INTO v_onum, v_amt, v_odate, v_cnum, v_snum;
  END LOOP;
  CLOSE top_orders_cursor;
END;
/
```

```
Top 5 Orders with the Highest Amounts:
ONUM    AMT     ODATE   CNUM    SNUM
-------------------------------------------
O3011   9891.88 06/10/1990      C2006   S1001
O3005   5160.45 10/04/1990      C2003   S1002
O3008   4723    05/10/1990      C2006   S1001
O3002   1900.1  10/03/1990      C2007   S1004
O3009   1713.23 04/10/1990      C2002   S1003

Statement processed.
```

To write a Cursor to display the list of customers who are living in San jose or London.

```
DECLARE
 CURSOR customer_cursor IS
  SELECT CNUM, CNAME, CITY, RATING, SNUM
  FROM CUSTOMER
  WHERE CITY IN ('San Jose', 'London');
 v_cnum CUSTOMER.CNUM%TYPE;
 v_cname CUSTOMER.CNAME%TYPE;
 v_city CUSTOMER.CITY%TYPE;
 v_rating CUSTOMER.RATING%TYPE;
 v_snum CUSTOMER.SNUM%TYPE;
BEGIN
 OPEN customer_cursor;
 FETCH customer_cursor INTO v_cnum, v_cname, v_city, v_rating, v_snum;
  DBMS_OUTPUT.PUT_LINE('List of Customers Living in San Jose or London:');
 DBMS_OUTPUT.PUT_LINE('CNUM' || CHR(9) || 'CNAME' || CHR(9) || 'CITY' || CHR(9) || 'RATING' || CHR(9) || 'SNUM');
 DBMS_OUTPUT.PUT_LINE('-------------------------------------------------------------');
  WHILE customer_cursor%FOUND LOOP
  DBMS_OUTPUT.PUT_LINE(v_cnum || CHR(9) || v_cname || CHR(9) || v_city || CHR(9) || v_rating || CHR(9) || v_snum);
  FETCH customer_cursor INTO v_cnum, v_cname, v_city, v_rating, v_snum;
 END LOOP;
  CLOSE customer_cursor;
END;
/
```

```
List of Customers Living in San Jose or London:
CNUM    CNAME   CITY     RATING  SNUM
-------------------------------------------------------------
C2001   Hardik  London   100     S1001
C2003   Kavish  San Jose         200     S1002
C2006   Vyomesh San Jose         300     S1007
C2005   Pratham London   100     S1001

Statement processed.
```

Calculate hra,da, gross and net by using PL/SQL program

```
DECLARE
  v_basic NUMBER := 15000; -- Replace with the desired basic salary
  v_hra NUMBER;
  v_da NUMBER;
  v_gross NUMBER;
  v_net NUMBER;
BEGIN
  -- Calculate HRA based on the given rules
  CASE
    WHEN v_basic = 15000 THEN
      v_hra := v_basic * 0.12;
      v_da := v_basic * 0.08;
    WHEN v_basic = 12000 THEN
      v_hra := v_basic * 0.10;
      v_da := v_basic * 0.06;
    WHEN v_basic = 9000 THEN
      v_hra := v_basic * 0.07;
       v_da := v_basic * 0.04;
    ELSE
      v_hra := v_basic * 0.05;
       v_da := 200;
  END CASE;
```

Basic	HRA	DA
15000	12%	8%
12000	10%	6%
9000	7%	4%
OTHERS	5%	200/-

```
Basic: 15000
HRA: 1800
DA: 1200
Gross: 18000
Net: 18000

Statement processed.
```

Function

- A function is similar to a procedure, except that it returns a single value.
- Functions can accept one, many, or no parameters, but a function must have a return clause in the executable section of the function.
- The datatype of the return value must be declared in the header of the function.

- The syntax for creating a function is as follows:

```
CREATE [OR REPLACE] FUNCTION function_name
        (parameter list)
        RETURN datatype
 IS/AS
 BEGIN
        <body>
        RETURN (return_value);
 END;
```

Function

The Following Function will fetch the name of the account for a given account number.

EXAMPLE:

```
Create or replace function account_det
 (account_num varchar2)
RETURN varchar2
IS
        acc_name varchar2(30);
BEGIN
        select name into acc_name from
    account where acc_no=account_num;
 RETURN acc_name;
END;
```

2 Ways to Execute the Function:

```
DECLARE
        acc_name VARCHAR2(30);
BEGIN
        acc_name :=
account_det(&account_num);
        DBMS_OUTPUT.PUT_LINE(acc_name);
END;
```

```
Select
account_det(acc_no)
from account;
```

Function

Write a function to accept employee number as parameter and return Basic +HRA together as single column

EXAMPLE:

```
CREATE OR REPLACE FUNCTION calculate_total_salary(p_emp_number NUMBER)
RETURN NUMBER
IS
   v_basic_salary NUMBER;
   v_hra NUMBER;
   v_total_salary NUMBER;
BEGIN
   -- Assuming you have a table named employees with columns emp_number, basic_salary,
and hra
   SELECT basic, hra
   INTO v_basic_salary, v_hra
   FROM employees
   WHERE emp_number = p_emp_number;

   v_total_salary := v_basic_salary + v_hra;

   RETURN v_total_salary;
END;
/
```

TO run: Select calculate_total_salary (emp_number) from account;

To run:

```
DECLARE
   v_total_salary NUMBER;
BEGIN
   v_total_salary := calculate_total_salary(123); -- Replace
123 with the desired employee number
   DBMS_OUTPUT.PUT_LINE('Total Salary: ' ||
v_total_salary);
END;
```

Trigger

Marwadi University

- A trigger is a 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.

Syntax of Triggers

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

Note:[REFERENCING OLD AS o NEW AS n] − This allows you to refer new and old values for various DML statements, such as INSERT, UPDATE, and DELETE.

Trigger

- The syntax for a dropping a Trigger is:

DROP TRIGGER *trigger_name ON tbl_name;*

- The syntax for a disabling a Trigger is:

ALTER TRIGGER *trigger_name* DISABLE;

- The syntax for a enabling a Trigger is:

ALTER TRIGGER *trigger_name* ENABLE;

Trigger

Creating Trigger

1.
CREATE TRIGGER deleted_details
BEFORE
DELETE on loan
FOR EACH ROW

2.
CREATE TRIGGER inserted_details
after
insert on transaction
FOR EACH ROW

Trigger

Creating Trigger

3.

```
CREATE TRIGGER updated_details
BEFORE
UPDATE on account
FOR EACH ROW when name='Patel Hiren';
```

Deleting Trigger

```
Drop trigger inserted_details on account;
```

Trigger

Understand and Implement Triggers.

1. Whenever order amount is updated and its value becomes more than 5000 a trigger has to be raised preventing the operation.

```
CREATE OR REPLACE TRIGGER prevent_high_order_amount
BEFORE UPDATE OF order_amount ON “order”
FOR EACH ROW
BEGIN
  IF :NEW.order_amount > 5000 THEN
    RAISE_APPLICATION_ERROR(-20001, 'Order amount cannot exceed 5000');
  END IF;
END;
```

Marwadi University

Thank You