# Practical 3

## PLSQL BASICS

**1. Write a PL/SQL block to take the number and string from user and display it.**

```
SQL> DECLARE
  2  num number;
  3  name varchar2(10);
  4  BEGIN
  5  num:=&num;
  6  name:='&name';
  7
  8  DBMS_OUTPUT.PUT_LINE('Number: '||num||', Name: '||name);
  9  END;
 10  /
Enter value for num: 10
old   5: num:=&num;
new   5: num:=10;
Enter value for name: Hammad
old   6: name:='&name';
new   6: name:=’Hammad’;
Number: 10, Name: Hammad

PL/SQL procedure successfully completed.
```

**2. Write a PL/SQL block to add two numbers.**

```
SQL> DECLARE
  2  num1 number;
  3  num2 number;
  4
  5  BEGIN
  6  num1:=&number1;
  7  num2:=&number2;
  8
  9  DBMS_OUTPUT.PUT_LINE('Addition: '||(num1+num2));
 10  END;
 11  /
Enter value for number1: 10
old   6: num1:=&number1;
new   6: num1:=10;
Enter value for number2: 5
old   7: num2:=&number2;
new   7: num2:=5;
Addition: 15

PL/SQL procedure successfully completed.
```

**3. Write a PL/SQL block to find the greatest among three numbers.**

```
DECLARE
a number(15);
b number(15);
```

```
c number(15);

BEGIN
a:=&a;
b:=&b;
c:=&c;

IF a>b AND a>c THEN
      DBMS_OUTPUT.put_line('A is greatest!');
ELSIF b>a AND b>c THEN
      DBMS_OUTPUT.put_line('B is greatest!');
ELSIF c>a AND c>b THEN
      DBMS_OUTPUT.put_line('C is greatest!');
ELSE
      DBMS_OUTPUT.put_line('ALL ARE EQUAL!');
END IF;
END;

SQL>  @ e:\hammaddbms\basic1_3.sql
 21  /
Enter value for a: 2
old   7: a:=&a;
new   7: a:=2;
Enter value for b: 3
old   8: b:=&b;
new   8: b:=3;
Enter value for c: 4
old   9: c:=&c;
new   9: c:=4;
C is greatest!

PL/SQL procedure successfully completed.
```

**4. Write a PL/SQL block to find out sum of first five numbers.**

```
set serveroutput ON;
DECLARE
i number(15);
s number(15);

BEGIN
s:=0;

FOR i IN 1..5
 LOOP
  s:=s+i;
 END LOOP;
      DBMS_OUTPUT.put_line('Sum of first five numbers = '||s);

END;
```

```
SQL> @ e:\hammaddbms\basic1_4.sql
 15  /
Sum of first five numbers =15


PL/SQL procedure successfully completed.
```

**5. Write a PL/SQL block to retrieve values**

```
set serveroutput ON;
DECLARE
name varchar2(100);

BEGIN
select bname into name from branch_02 where city='BANGLORE';
DBMS_OUTPUT.put_line(name);
END;

SQL> @ e:\hammaddbms\basic1_5.sql
  8  /
M.G.ROAD

PL/SQL procedure successfully completed.
```

## PLSQL CURSORS

**1. Display the depositor names and amount of virar branch using cursor.**

```
SET SERVEROUTPUT ON;
DECLARE
cursor c1 is select CNAME cn, AMOUNT am from deposit_02 where BNAME =
'VIRAR';
z c1%rowtype;
BEGIN
open c1;
fetch c1 into z;
while(c1%found)loop
    dbms_output.put_line(z.cn ||' '||z.am);
    fetch c1 into z;
end loop;
```

```
close c1;
end;
/
SQL> @ E:\HammadDBMS\PLSQL_CURSOR\cursor_1.sql
 13  /
SHIVANI 1000

PL/SQL procedure successfully completed.
```

**2. Display the name and amount of virar branch using parametric cursor.**

```
SET SERVEROUTPUT ON;
DECLARE
cursor c1(branch varchar) is select CNAME cn, AMOUNT am from deposit_02
where BNAME = branch;
z c1%rowtype;
BEGIN
open c1('VIRAR');
fetch c1 into z;
while(c1%found)loop
    dbms_output.put_line(z.cn ||' '||z.am);
    fetch c1 into z;
end loop;
close c1;
end;
/

SHIVANI 1000

PL/SQL procedure successfully completed.
```

**3. Display total number of rows of a customer table using for loop.**

```
SET SERVEROUTPUT ON;
DECLARE
cursor c1 is select * from customer_02;
z c1%rowtype;
counter number;
BEGIN
counter := 0;
for z in c1 loop
    counter:=counter+1;
end loop;
```

```
dbms_output.put_line('Count = '||counter);
end;
/
SQL> @ E:\HammadDBMS\PLSQL_CURSOR\cursor_3.sql
 16  /
Count = 10
```

**4. Display total number of rows of a customer table using while loop.**

```
SET SERVEROUTPUT ON;
DECLARE
cursor c1 is select * from customer_02;
z c1%rowtype;
counter number;
BEGIN
counter := 0;
open c1;
fetch c1 into z;
while(c1%found)loop
    counter:=counter+1;
    fetch c1 into z;
end loop;
dbms_output.put_line('Count = '||counter);
close c1;
end;
/
Count = 10

PL/SQL procedure successfully completed.
```

**5. Display total number of rows of a customer table using LOOP..END LOOP and %NOTFOUND.**

```
SET SERVEROUTPUT ON;
DECLARE
cursor c1 is select * from customer_02;
z c1%rowtype;
counter number;
BEGIN
counter := 0;
open c1;
fetch c1 into z;
loop
    counter:=counter+1;
```

```
    fetch c1 into z;
    EXIT WHEN c1%NOTFOUND;
end loop;
dbms_output.put_line('Count = '||counter);
close c1;
end;
/
Count = 10

PL/SQL procedure successfully completed.
```

**6. Diplay total amount of the depositors of virar branch.**

```
SET SERVEROUTPUT ON;
DECLARE
cursor c1 is select amount am from deposit_02 where bname = 'VIRAR';
z c1%rowtype;
sumamount number;
BEGIN
sumamount := 0;
open c1;
fetch c1 into z;
while(c1%found)loop
    sumamount:=sumamount+z.am;
    fetch c1 into z;
end loop;
dbms_output.put_line('Total Amount = '||sumamount);
close c1;
end;
/
Total Amount = 28700

PL/SQL procedure successfully completed.
```

**7. Calculate and display depositor name having forth maximum amount.**

```
SET SERVEROUTPUT ON;
DECLARE
cursor c1 is select cname nm from deposit_02 order by amount desc ;
z c1%rowtype;
BEGIN
open c1;
fetch c1 into z;
```

```
while(c1%found)loop
    if(c1%rowcount = 4)then
        dbms_output.put_line(z.nm);
    end if;
    fetch c1 into z;
end loop;
close c1;
end;
/
```

```
MEHUL

PL/SQL procedure successfully completed.
```

## PLSQL CURSORS PRACTICE

1. **Write a cursor to insert first five highest marks of students including their name and id from student table to temp table and display it**

```
SET SERVEROUTPUT ON;
DECLARE
cursor c1 is select * from STUDENT_P_02 ORDER BY MARKS DESC;
z c1%rowtype;
```

```
c NUMBER;
BEGIN
open c1;
c := 0;
fetch c1 into z;
while(c1%found)loop
    INSERT INTO TEMP_02 VALUES (z.SID,z.SNAME,z.MARKS);
    c := c + 1;
    EXIT WHEN c = 5;
    fetch c1 into z;
end loop;
close c1;
end;
/
PL/SQL procedure successfully completed.
```

| SID | SNAME | MARKS |
|---|---|---|
| 1 | Nishita | 80 |
| 2 | Hammad | 80 |
| 3 | Maulika | 80 |
| 4 | Diksha | 80 |
| 5 | Riya | 80 |
| 6 | Datta | 80 |
| 7 | Devendra | 80 |
| 8 | Siddhesh | 80 |
| 9 | Preeti | 80 |
| 11 | Anuradha | 80 |

10 rows selected.

```
SQL> select * from temp_student_22;
```

| SID | SNAME | MARKS |
|---|---|---|
| 1 | Nishita | 80 |
| 2 | Hammad | 80 |
| 3 | Maulika | 80 |
| 4 | Diksha | 80 |
| 11 | Anuradha | 80 |

**2. Write a PL/SQL block to increase the amount of a depositor whose branch is NAGPUR.**

- **a. The amount of increase is 20% for depositors having amount less than 10000 and 12% for depositors having amount less than 5000.**
- **b. Use a cursor with FOR UPDATE clause.**
- **c. Update the amount with a WHERE CURRENT OF clause in a cursor FOR loop (cursor FOR loop problem).**

```
SQL> DECLARE
```

```
  2  CURSOR cur_nagpur_customers IS SELECT * FROM deposit_02 WHERE
bname IN (SELECT bname FROM branch_02 WHERE city='NAGPUR') FOR UPDATE
OF amount nowait;
  3
  4  BEGIN
  5  FOR x IN cur_nagpur_customers LOOP
  6  IF x.amount<5000 THEN
  7  UPDATE deposit_02 SET amount=(amount+(amount*0.12)) WHERE CURRENT
OF cur_nagpur_customers;
  8  ELSIF x.amount<10000 THEN
  9  UPDATE deposit_02 SET amount=(amount+(amount*0.2)) WHERE CURRENT
OF cur_nagpur_customers;
 10  END IF;
 11  END LOOP;
 12  END;
 13  /

PL/SQL procedure successfully completed.

SQL>
SQL> select * from deposit_02;

ACTNO CNAME              BNAME                  AMOUNT ADATE
----- ------------------ ------------------ ---------- ---------
100   ANIL               VRCE                     1120 01-MAR-95
101   SUNIL              AJNI                     6000 04-JAN-96
102   MEHUL              KAROLBAGH                3500 17-NOV-95
104   MADHURI            CHANDNI                  1200 17-DEC-95
105   PRAMOD             M.G.ROAD                 3000 27-MAR-96
106   SANDIP             ANDHERI                  2000 31-MAR-96
107   SHIVANI            VIRAR                    1000 05-SEP-95
108   KRANTI             NEHRU PLACE              5000 02-JUL-95
109   NAREN              POWAI                    7000 10-AUG-95

9 rows selected.

SQL> rollback;

Rollback complete.
```

## TRIGGERS

1. **Create a trigger on emp table that does not allow salary to be less than 10000.**

```
SET SERVEROUTPUT ON;
CREATE OR REPLACE TRIGGER EMP_1
AFTER INSERT ON EMP_02
```

```
FOR EACH ROW

DECLARE

BEGIN
IF(:new.salary<10000) THEN
    raise_application_error(-20001,'Salary cant be less than 10000');
END IF;
END;
/

Trigger created.

SQL> INSERT INTO EMP VALUES(1,'HAMMAD',3000);
INSERT INTO EMP VALUES(1,'HAMMAD',3000)
            *
ERROR at line 1:
ORA-20001: Salary cant be less than 10000
ORA-06512: at "SYSTEM.EMP_1", line 5
ORA-04088: error during execution of trigger 'SYSTEM.EMP_1'
```

**2. Create a trigger on emp table that does not allow empid to be more than 2 digits.**

```
SET SERVEROUTPUT ON;
CREATE OR REPLACE TRIGGER EMP_2
AFTER INSERT ON EMP_02
FOR EACH ROW

DECLARE

BEGIN
IF(:new.EMPID>100) THEN
    raise_application_error(-20002,'EMP ID CANNOT BE MORE THAN 2
DIGITS');
END IF;
END;
/


Trigger created.
SQL> INSERT INTO EMP VALUES(100,'HAMMAD',30000);
INSERT INTO EMP VALUES(100,'HAMMAD',30000)
            *
ERROR at line 1:
ORA-20002: EMP ID CANNOT BE MORE THAN 2 DIGITS
ORA-06512: at "SYSTEM.EMP_2", line 5
ORA-04088: error during execution of trigger 'SYSTEM.EMP_2'


SQL> INSERT INTO EMP VALUES(1,'HAMMAD',95000);
```

```
1 row created.

SQL> SELECT * FROM EMP;

     EMPID EMPNAME                    SALARY
---------- -------------------- ----------
         1 HAMMAD                    95000
```

**3. Create a trigger which does not allow DML operations on emp table if the username is System.**

```
CREATE OR REPLACE TRIGGER trigger_no_system_allowed BEFORE INSERT OR
UPDATE OR DELETE ON emp_02 FOR EACH ROW
   DECLARE
   user_name varchar2(10);
   BEGIN
   SELECT user INTO user_name FROM DUAL;
   IF (user_name='System') THEN
   RAISE_APPLICATION_ERROR(-20000,'System IS NOT ALLOWED!');
   END IF;
   END;
   /

Trigger created.

SQL> insert into emp_02 values(10,’ABC’,3333);
insert into emp_02 values(10,'ABC',3333)
           *
ERROR at line 1:
ORA-20000: SYSTEM IS NOT ALLOWED!
ORA-06512: at "SYSTEM.TRIGGER_NO_SYSTEM_ALLOWED", line 6
ORA-04088: error during execution of trigger
'SYSTEM.TRIGGER_NO_SYSTEM_ALLOWED'


SQL> drop trigger trigger_no_system_allowed;

Trigger dropped.
```

**4. Create a trigger that allows no DML operations on emp table to be performed on any weekdays but allow insertion on Sunday**

```
SQL> CREATE OR REPLACE TRIGGER trigger_specified_days BEFORE INSERT OR
UPDATE OR DELETE ON emp_02 FOR EACH ROW
   DECLARE
   day_name varchar2(10);
   BEGIN
   SELECT TO_CHAR(SYSDATE,'DAY') INTO day_name FROM DUAL;
   IF (day_name='SUNDAY') THEN
   IF UPDATING OR DELETING THEN
   RAISE_APPLICATION_ERROR(-20000,'PLEASE READ SUNDAY CURRICULUM.');
   END IF;
```

```
 10  ELSIF (day_name='MONDAY') OR (day_name='TUESDAY') OR
(day_name='FRIDAY') OR (day_name='WEDNESDAY') OR (day_name='THURSDAY')
THEN
 11  IF UPDATING OR DELETING OR INSERTING THEN
 12  RAISE_APPLICATION_ERROR(-20000,'PLEASE READ WEEKDAY CURRICULUM.');
 13  END IF;
 14  END IF;
 15  END;
 16  /

Trigger created.

SQL>
SQL>
SQL>
SQL> --Dummy test
SQL> insert into emp_02 values(10,'ABC',3333);

1 row created.

SQL> select to_char(sysdate,'day-month-year') from dual;

TO_CHAR(SYSDATE,'DAY-MONTH-YEAR')
--------------------------------------------------------------------------
---------
-saturday -september-twenty nineteen
```

**5. Create a trigger on tempfees when updation is performed then the old values of tempfees are copied into final fees.**

```
SQL> CREATE OR REPLACE TRIGGER trigger_pass_value_to_next AFTER UPDATE
ON tempfees_02 FOR EACH ROW
  2  BEGIN
  3  INSERT INTO finalfees_02 VALUES(:old.amount);
  4  END;
  5  /

Trigger created.

SQL>
SQL> insert into tempfees_02 values(10000);

1 row created.

SQL> select * from tempfees_02;

    AMOUNT
----------
     10000

SQL> select * from finalfees_02;

no rows selected

SQL> update tempfees_02 set amount=20000 where amount=10000;
```

```
1 row updated.

SQL> update tempfees_02 set amount=30000 where amount=20000;

1 row updated.

SQL> select * from tempfees_02;

    AMOUNT
----------
     30000

SQL> select * from finalfees_02;

    AMOUNT
----------
     10000
     20000
```

## TRIGGERS PRACTICE

**1. Write a trigger that performs cascading update.**

```
CREATE OR REPLACE TRIGGER UPDATE_ON_TRIGGER_1
 AFTER UPDATE OF A ON Parent_02 FOR EACH ROW
BEGIN
  UPDATE Child_02
    SET A = :new.A
    WHERE A = :old.A;
END;
/

Trigger created.

SQL> INSERT INTO Parent_02 VALUES(3,6);

1 row created.

SQL> SELECT * FROM Child_02;

         A          B
---------- ----------
         1         10
         2          7

SQL> SELECT * FROM Parent_02;

         A          B
---------- ----------
         1          2
         2          4
```

```
         3          6

SQL> UPDATE Parent_02 SET A =5 WHERE A=1;

1 row updated.

SQL> SELECT * FROM PARENT_02;

         A          B
---------- ----------
         5          2
         2          4
         3          6

SQL> SELECT * FROM CHILD_02;

         A          B
---------- ----------
         5         10
         2          7
```

**2. Write a trigger that performs reverse cascading update.**

```
CREATE OR REPLACE TRIGGER UPDATE_ON_TRIGGER_2
 AFTER UPDATE OF A ON Child_02 FOR EACH ROW
BEGIN
  UPDATE Parent_02
    SET A = :new.A
    WHERE A = :old.A;
END;
/


Trigger created.

SQL> update child_02 set a=1 where a=10;

1 row updated.

SQL> select * from parent_02;

         A          B
---------- ----------
         1          2
         2          4

SQL> select * from child_22;

         A          B
---------- ----------
         1         10
         2          7
```

**3. Write a trigger that performs cascading delete.**

```
CREATE OR REPLACE TRIGGER DELETE_ON_TRIGGER
AFTER DELETE ON Parent_02
REFERENCING OLD AS OLDROW
FOR EACH ROW
BEGIN
  DELETE FROM Child_02 WHERE Child_02.A = :OLDROW.A;
END;
/


Trigger created.
SQL> DELETE FROM PARENT_02 WHERE A = 5;

1 row deleted.

SQL> SELECT * FROM CHILD_02;

         A          B
---------- ----------
         2          7

SQL> SELECT * FROM PARENT_02;

         A          B
---------- ----------
         2          4
         3          6

SQL> SPOOL OFF
```

## RECORDS

**1. Take the record values from user and store it in the college table.**

```
declare
type COLLEGE_REC is RECORD (COLLEGEID number,NAME varchar2(25), ADDRESS
varchar2(50));
x COLLEGE_REC;
begin
x.COLLEGEID := &COLLEGEID;
x.NAME := '&NAME';
X.ADDRESS := '&ADDRESS';
INSERT INTO COLLEGE_02 VALUES(X.COLLEGEID,X.NAME,X.ADDRESS);
end;
/
```

```
Enter value for collegeid: 1
old   5: x.COLLEGEID := &COLLEGEID;
new   5: x.COLLEGEID := 1;
Enter value for name: SPIT
old   6: x.NAME := '&NAME';
new   6: x.NAME := 'SPIT';
Enter value for address: MUNSHI NAGAR
old   7: X.ADDRESS := '&ADDRESS';
new   7: X.ADDRESS := 'MUNSHI NAGAR';

PL/SQL procedure successfully completed.

SQL> SELECT * FROM COLLEGE_02;

 COLLEGEID NAME
---------- -------------------------
ADDRESS
--------------------------------------------------
         1 SPIT
MUNSHI NAGAR
```

**2. Update the address of SPIT college to Andheri in college table using record.**

```
declare
type college_rec is RECORD (ADDRESS varchar2(50));
x college_rec;
begin
x.address := '&address';
UPDATE COLLEGE_02
SET address = x.address
WHERE name='SPIT';
end;
/

Enter value for address: ANDHERI
old   5: x.address := '&address';
new   5: x.address := 'ANDHERI';

PL/SQL procedure successfully completed.

SQL> SELECT * FROM COLLEGE_02;

 COLLEGEID NAME
---------- -------------------------
ADDRESS
--------------------------------------------------
         1 SPIT
```

ANDHERI

**3. Display all college details from college table using record.**

```
SET SERVEROUTPUT ON;
DECLARE
type rec is RECORD(COLLEGEID number, NAME varchar2(25), ADDRESS
varchar2(50));
x rec;
CURSOR c1 IS SELECT * FROM COLLEGE_02;
BEGIN
FOR x IN c1 LOOP
    DBMS_OUTPUT.put_line('College ID: ' || x.collegeid);
    DBMS_OUTPUT.put_line(' Name: ' || x.name );
    DBMS_OUTPUT.put_line(' Address: ' || x.address);
END LOOP;
END;
/
 SQL>  @ E:\HammadDBMS\PLSQL_RECORDS\RECORD_03.SQL
College ID: 1
Name: SPIT
Address: ANDHERI

PL/SQL procedure successfully completed.

SQL> SPOOL OFF;
```

## EXCEPTIONS

**4. Perform following query using PL/SQL for above database. Select (expectedincome/nvl(netincome,0)) into x from batch where Bcode=103; It will give you divide by zero error. Write a PL/SQL block to handle this exception.**

```
SET SERVEROUTPUT ON;
DECLARE
X NUMBER;
BEGIN
Select (expectedincome/nvl(netincome,0)) into x from BATCH_02 where
BCODE=103;
EXCEPTION
   WHEN ZERO_DIVIDE THEN
      dbms_output.put_line('DIVIDING BY ZERO PLEASE CHECK THE VALUES
AGAIN');
END;
/

CLEAR SCREEN;
SQL> CREATE TABLE BATCH_02(BCODE NUMBER, EXPECTEDINCOME NUMBER,
NETINCOME NUMBER);

Table created.

SQL> INSERT INTO BATCH_02
VALUES(&BCODE,&EXPECTEDINCOME,&NETINCOME);
Enter value for bcode: 101
Enter value for expectedincome: 10000
Enter value for netincome: 1233
old   1: INSERT INTO BATCH_02
VALUES(&BCODE,&EXPECTEDINCOME,&NETINCOME)
new   1: INSERT INTO BATCH_02 VALUES(101,10000,1233)

1 row created.

SQL> /
Enter value for bcode: 102
Enter value for expectedincome: 20000
Enter value for netincome: 1111
old   1: INSERT INTO BATCH_02
VALUES(&BCODE,&EXPECTEDINCOME,&NETINCOME)
new   1: INSERT INTO BATCH_02 VALUES(102,20000,1111)

1 row created.

SQL> /
```

```
Enter value for bcode: 103
Enter value for expectedincome: 30000
Enter value for netincome: NULL
old   1: INSERT INTO BATCH_02
VALUES(&BCODE,&EXPECTEDINCOME,&NETINCOME)
new   1: INSERT INTO BATCH_02 VALUES(103,30000,NULL)

1 row created.

SQL> /
Enter value for bcode: 104
Enter value for expectedincome: 40000
Enter value for netincome: 1000
old   1: INSERT INTO BATCH_02
VALUES(&BCODE,&EXPECTEDINCOME,&NETINCOME)
new   1: INSERT INTO BATCH_02 VALUES(104,40000,1000)

1 row created.

SQL> /
Enter value for bcode: 105
Enter value for expectedincome: 50000
Enter value for netincome: NULL
old   1: INSERT INTO BATCH_02
VALUES(&BCODE,&EXPECTEDINCOME,&NETINCOME)
new   1: INSERT INTO BATCH_02 VALUES(105,50000,NULL)

1 row created.

SQL> SELECT * FROM BATCH_02;

     BCODE EXPECTEDINCOME  NETINCOME
---------- -------------- ----------
       101          10000       1233
       102          20000       1111
       103          30000
       104          40000       1000
       105          50000


PL/SQL procedure successfully completed.

SQL> @ E:\HammadDBMS\PLSQL_EXCEPTION\EXCEP_01_E.SQL;
DIVIDING BY ZERO PLEASE CHECK THE VALUES AGAIN

PL/SQL procedure successfully completed.
```

**5. Write a PL/SQL block to handle the user defined exception on emp table if the newly inserted salary is less than 10000.**

```
SET SERVEROUTPUT ON;
CREATE OR REPLACE TRIGGER EMP_TRIG
BEFORE INSERT ON EMP_02
FOR EACH ROW
DECLARE
LESS_SALARY EXCEPTION;
BEGIN
IF(:new.SALARY<10000)THEN
   RAISE LESS_SALARY;
END IF;
EXCEPTION
   WHEN LESS_SALARY THEN
      dbms_output.put_line('SALARY SHOULD BE GREATER THAN 10000');
END;
/


SQL> INSERT INTO EMP_02 VALUES(2,'NISHITA',5000);
SALARY SHOULD BE GREATER THAN 10000

1 row created.

SQL> COMMIT;

Commit complete.

SQL> SPOOL OFF;
```