*** EXPERIMENT NO: 05 ***

Aim : To write and execute PL/SQL blocks (with exception handling) and Cursors using Oracle 11g.

Problem Statement : Establish the database relation EMPLOYEE and populate it with sample records. The logical schema of EMPLOYEE table is –

EMPLOYEE (EID, FNAME, LNAME, BIRTHDATE, GENDER, SSN, HIREDATE, SALARY, DEPARTMENT, DESIGNATION)

Author : Bhavesh Kewalramani

Roll No : 025 [5A]

Date : 08-September-2021

****

QUERY-01: Write SQL code to create and execute an anonymous PL/SQL block that will

insert 5 tuples into EXAM. Ensure to commit the populated records.

Test the insertion in EXAM by displaying its contents.

/*

Create a table EMPP (contains no records at creation) that includes EID, ENAME (column combining FNAME and LNAME with embedded blank), HIREDATE, DESIGNATION and SALARY from EMPLOYEE table. Enforce entity integrity constraints on EID. Verify table creation, contents and constraints.

***/**

****

```
BEGIN
      FOR I IN 1001 .. 1005 LOOP
              INSERT INTO EXAM (UROLL, COURSE, EXAMDT)
              VALUES (TO_NUMBER(I),'DBMS','20-SEP-2021');
              DBMS_OUTPUT.PUT_LINE('INSERTED RECORD FOR UROLL :=
'||I);
      END LOOP;
COMMIT;
END;
/
```

```
INSERTED RECORD FOR UROLL := 1001
INSERTED RECORD FOR UROLL := 1002
INSERTED RECORD FOR UROLL := 1003
INSERTED RECORD FOR UROLL := 1004
INSERTED RECORD FOR UROLL := 1005

PL/SQL procedure successfully completed.
```

```
             SELECT *
             FROM EXAM;
```

```
     UROLL COUR EXAMDT
---------- ---- ---------
      1001 DBMS 20-SEP-21
      1002 DBMS 20-SEP-21
      1003 DBMS 20-SEP-21
      1004 DBMS 20-SEP-21
      1005 DBMS 20-SEP-21

5 rows selected.
```

```
CREATE TABLE EMPP
AS (SELECT ENO AS EID,FNAME||' '||LNAME AS ENAME,HIREDATE,DESIGNATION,SALARY
    FROM EMPLOYEE
    WHERE 1=2);
```

```
Table created.
```

```
ALTER TABLE EMPP
ADD
CONSTRAINT EMPP_PK_EID PRIMARY KEY (EID);
```

```
Table altered.
```

```
SELECT CONSTRAINT_NAME,CONSTRAINT_TYPE
FROM USER_CONSTRAINTS
WHERE UPPER(TABLE_NAME)='EMPP';
```

```
CONSTRAINT_NAME                C
------------------------------ -
SYS_C007770                    C
SYS_C007769                    C
SYS_C007768                    C
SYS_C007767                    C
EMPP_PK_EID                    P

5 rows selected.
```

```
      SELECT *
      FROM EMPP;
```

```
no rows selected
```

```
      DESC EMPP
```

```
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 EID                                       NOT NULL NUMBER(4)
 ENAME                                              VARCHAR2(21)
 HIREDATE                                  NOT NULL DATE
 DESIGNATION                               NOT NULL VARCHAR2(15)
 SALARY                                    NOT NULL NUMBER(8,2)
```

****

QUERY-02: Write SQL code to create and execute an anonymous PL/SQL block that
will use %TYPE variables to populate the EMPP table with corresponding tuples
in EMPLOYEE table.

```
/*
      Create a table MENTEE (contains no records at creation) that includes
      Staff Number, Staff Name, Student Name (column combining FNAME and
      LNAME
      with embedded blank), Roll Number and registration date from STUDENT
      and
      STAFF tables. Enforce entity integrity constraints on combination of
      Staff Number and Roll Number. Verify table creation, contents and
      constraints.
*/
********************************************************************************

DECLARE
     I EMPP.EID%TYPE;
     J EMPP.ENAME%TYPE;
     K EMPP.HIREDATE%TYPE;
     L EMPP.SALARY%TYPE;
     M EMPP.DESIGNATION%TYPE;
BEGIN
     FOR I IN 7101..7117 LOOP
                 SELECT FNAME||' '||LNAME AS ENAME,HIREDATE,SALARY,DESIGNATION
                 INTO J,K,L,M
                 FROM EMPLOYEE
                 WHERE ENO=I;
                 INSERT INTO EMPP(EID,ENAME,HIREDATE,SALARY,DESIGNATION)
                 VALUES(I,J,K,L,M);
                 DBMS_OUTPUT.PUT_LINE('SUCCESSFULLY INSERTED
                                                         RECORD OF EID := '||I);
     END LOOP;
END;
/

SUCCESSFULLY INSERTED RECORD OF EID := 7101
SUCCESSFULLY INSERTED RECORD OF EID := 7102
SUCCESSFULLY INSERTED RECORD OF EID := 7103
SUCCESSFULLY INSERTED RECORD OF EID := 7104
SUCCESSFULLY INSERTED RECORD OF EID := 7105
SUCCESSFULLY INSERTED RECORD OF EID := 7106
SUCCESSFULLY INSERTED RECORD OF EID := 7107
SUCCESSFULLY INSERTED RECORD OF EID := 7108
SUCCESSFULLY INSERTED RECORD OF EID := 7109
SUCCESSFULLY INSERTED RECORD OF EID := 7110
SUCCESSFULLY INSERTED RECORD OF EID := 7111
SUCCESSFULLY INSERTED RECORD OF EID := 7112
SUCCESSFULLY INSERTED RECORD OF EID := 7113
SUCCESSFULLY INSERTED RECORD OF EID := 7114
SUCCESSFULLY INSERTED RECORD OF EID := 7115
SUCCESSFULLY INSERTED RECORD OF EID := 7116
SUCCESSFULLY INSERTED RECORD OF EID := 7117
```

```
PL/SQL procedure successfully completed.

      CREATE TABLE MENTEE AS
      (SELECT S1.SID AS STAFF_NUMBER,
              S1.NAME AS STAFF_NAME,
              S2.FNAME||' '||S2.LNAME AS STUDENT_NAME,
              S2.ROLL AS ROLL,
              S2.REG_DT AS REG_DT
      FROM STAFF S1 JOIN STUDENT S2
      ON S1.SID = S2.ADVISOR AND 1=2);

Table created.

      ALTER TABLE MENTEE
      ADD
      PRIMARY KEY (STAFF_NUMBER,ROLL);

Table altered.

      SELECT *
      FROM MENTEE;

no rows selected

      SELECT CONSTRAINT_NAME,CONSTRAINT_TYPE
      FROM USER_CONSTRAINTS
      WHERE UPPER(TABLE_NAME)='MENTEE';

CONSTRAINT_NAME                C
------------------------------ -
SYS_C007789                    C
SYS_C007791                    C
SYS_C007790                    C
SYS_C007792                    P

4 rows selected.
```

****

QUERY-03: Write SQL code to create and execute an anonymous PL/SQL block that will use %ROWTYPE variables to populate the MENTEE table with corresponding tuples from Academic Schema.
****

```
      DECLARE
           M_REC MENTEE%ROWTYPE;
           CURSOR M_REC_CUR IS
                   SELECT S1.SID,S1.NAME,S2.FNAME||' '||S2.LNAME,S2.ROLL,S2.REG_DT
                   FROM STAFF S1,STUDENT S2
                   WHERE S1.SID = S2.ADVISOR;
      BEGIN
           OPEN M_REC_CUR;
           LOOP
                   FETCH M_REC_CUR INTO M_REC;
                   EXIT WHEN M_REC_CUR%NOTFOUND;
                   INSERT INTO MENTEE VALUES M_REC;
           END LOOP;
           DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT||
      END;
      /
```

```
PL/SQL procedure successfully completed.
```

****

QUERY-04: Write SQL code to create and execute an anonymous PL/SQL block that will display the contents of MENTEE table without using declared variables. You should format the output using RPAD() and/ or LPAD(), while including proper headers in the result.
****

```
BEGIN
          DBMS_OUTPUT.PUT_LINE('  STAFF_NUMBER
                                STAFF_NAME
                                      STUDENT_NAME   ROLL  REG_DT  ');
          DBMS_OUTPUT.PUT_LINE('==============
                                =========================
                                      ============================   ====
                                                                ========');
          FOR CUR IN (SELECT * FROM MENTEE)
          LOOP
            DBMS_OUTPUT.PUT_LINE(RPAD(CUR.STAFF_NUMBER,14)||'  '||
                                 RPAD(CUR.STAFF_NAME,25)||'  '||
                                 RPAD(CUR.STUDENT_NAME,28)||'  '||
                                RPAD(CUR.ROLL,3)||'  '||RPAD(CUR.REG_DT,9));
           END LOOP;
END;
/
```

```
STAFF_NUMBER                 STAFF_NAME                  STUDENT_NAME    ROLL
REG_DT
==============  =========================  ============================  ====
=========
101             Kamalkant Marathe          Afra Sayed                    1    20-
JUL-18
104             Aasawari Deodhar           Akansha Wasalu                2    20-
JUL-18
108             Jasmine Arora              Anjali Rajendran              3    19-
JUL-18
109             Vallabh Pai                Aradhita Menghal              4    07-
JUL-18
101             Kamalkant Marathe          Ritul Deshmukh                11   18-
JUL-18
104             Aasawari Deodhar           Sakshi Nema                   12   07-
JUL-18
108             Jasmine Arora              Shreya Agnihotri              13   07-
JUL-18
109             Vallabh Pai                Shrishti Shukla               14   19-
JUL-18
101             Kamalkant Marathe          Aayush Muley                  31   19-
JUL-18
104             Aasawari Deodhar           Abhishek Chohan               32   07-
JUL-18
108             Jasmine Arora              Adesh Kotgirwar               33   20-
JUL-18
109             Vallabh Pai                Adhney Nawghare               34   08-
AUG-18
101             Kamalkant Marathe          Ayush Gupta                   41   12-
JUL-18
104             Aasawari Deodhar           Chaitanya Kapre               42   25-
JUL-18
108             Jasmine Arora              Dev Paliwal                   43   21-
JUL-18
109             Vallabh Pai                Gaurav Shukla                 44   17-
JUL-18
109             Vallabh Pai                Keshubh Sharma                53   20-
JUL-18
```

108	Jasmine Arora	Kunal Thorane	54	08-AUG-18
104	Aasawari Deodhar	Mehul Khandhadiya	55	19-JUL-18
101	Kamalkant Marathe	Nikhil Tiwari	56	04-JUL-18
104	Aasawari Deodhar	Rishikesh Kale	63	07-JUL-18
108	Jasmine Arora	Ritik Parashar	64	19-JUL-18
101	Kamalkant Marathe	Rohit Chandani	65	08-AUG-18
109	Vallabh Pai	Shubham Jha	78	12-JUL-18
108	Jasmine Arora	Yaman Kushwah	79	17-JUL-18
104	Aasawari Deodhar	Yash Bhageriya	80	19-JUL-18
109	Vallabh Pai	Renuka Soni	30	25-JUL-16
108	Jasmine Arora	Mayank Rangari	87	25-JUL-16
102	Adishesh Vidyarthi	Ketki Fadnavis	5	14-JUL-18
110	Harmeet Kullar	Lalita Sharma	6	10-JUL-18
102	Adishesh Vidyarthi	Simran Baheti	15	20-JUL-18
110	Harmeet Kullar	Urvi Negi	16	19-JUL-18
102	Adishesh Vidyarthi	Akshat Chandak	35	20-JUL-18
110	Harmeet Kullar	Amey Chole	36	08-AUG-18
110	Harmeet Kullar	Gursewak Virdi	45	07-JUL-18
102	Adishesh Vidyarthi	Saurabh Khandagale	46	10-AUG-19
102	Adishesh Vidyarthi	Paritosh Dandekar	57	14-JUL-18
110	Harmeet Kullar	Pavankumar Gupta	58	03-JUL-18
110	Harmeet Kullar	Rushil Parikh	71	07-JUL-18
102	Adishesh Vidyarthi	Sankalp Pandey	72	07-JUL-18

102	Adishesh Vidyarthi	Yash Daware	81	20-JUL-18
110	Harmeet Kullar	Yash Roy	82	07-JUL-18
110	Harmeet Kullar	Love Sharnagat	68	25-JUL-17
103	Manishi Singh	Muskan Gupta	7	19-JUL-18
106	Deo Narayan Mishra	Prateeksha Devikar	8	13-JUL-18
106	Deo Narayan Mishra	Deepali Pathe	17	10-AUG-19
103	Manishi Singh	Prachi Bhanuse	18	11-AUG-19
103	Manishi Singh	Amit Ray	37	20-JUL-18
106	Deo Narayan Mishra	Aryan Pandharipande	38	07-JUL-18
106	Deo Narayan Mishra	Ganesh Thakur	47	22-AUG-19
103	Manishi Singh	Manishkumar Pardhi	48	23-AUG-19
103	Manishi Singh	Rahul Agrawal	59	16-JUL-18
106	Deo Narayan Mishra	Rajat Chandak	60	20-JUL-18
103	Manishi Singh	Saurabh Sushir	73	07-JUL-18
106	Deo Narayan Mishra	Shardul Nimbalkar	74	28-JUL-17
106	Deo Narayan Mishra	Yash Dhamecha	83	21-JUL-18
103	Manishi Singh	Yash Jain	84	03-JUL-18
103	Manishi Singh	Anujesh Soni	67	25-JUL-17
105	Geetika Goenka	Priyal Taori	9	19-JUL-18
107	Sanjeev Bamireddy	Rashi Chouksey	10	08-AUG-18
107	Sanjeev Bamireddy	Siddhi Tripathi	19	31-AUG-19
105	Geetika Goenka	Atharva Uplanchiwar	39	07-JUL-18
107	Sanjeev Bamireddy	Atharva Paliwal	40	20-JUL-18

```
105             Geetika Goenka            Harsh Karwa                    51   11-
JUL-18
107             Sanjeev Bamireddy         Jayesh Kapse                   52   08-
AUG-18
107             Sanjeev Bamireddy         Ram Agrawal                    61   19-
JUL-18
105             Geetika Goenka            Raunak Khandelwal              62   19-
JUL-18
105             Geetika Goenka            Shashank Tapas                 75   07-
JUL-18
107             Sanjeev Bamireddy         Shivam Bagadia                 76   20-
JUL-18
105             Geetika Goenka            Shreyas Nemani                 77   20-
JUL-18
105             Geetika Goenka            Yogesh Siral                   85   21-
JUL-18
107             Sanjeev Bamireddy         Shapath Pandey                 86   27-
JUL-17
107             Sanjeev Bamireddy         Ayush Singh                    66   27-
JUL-17
109             Vallabh Pai               Naveen Namjoshi                88   14-
AUG-19
110             Harmeet Kullar            Tushar Tipnis                  89   14-
AUG-19

PL/SQL procedure successfully completed.
```

****

QUERY-05: Write SQL code to create and execute an anonymous PL/SQL block that will display the system date. Use exception (exception VALUE_ERROR) to check if the variable holding the system date is large enough in size.

Re-execute the block with appropriate modification to test the exception.

****

```
DECLARE
      DT DATE := SYSDATE;
BEGIN
      DBMS_OUTPUT.PUT_LINE('DATE');
      DBMS_OUTPUT.PUT_LINE('=========');
      DBMS_OUTPUT.PUT_LINE(TO_CHAR(DT));
END;
/
```

```
DATE
=========
16-SEP-21

PL/SQL procedure successfully completed.
```

```
      DECLARE
            DT DATE := SYSDATE;
      BEGIN
            DBMS_OUTPUT.PUT_LINE('DATE');
            DBMS_OUTPUT.PUT_LINE('=========');
            DBMS_OUTPUT.PUT_LINE(TO_CHAR(DT));
      EXCEPTION
            WHEN VALUE_ERROR THEN
                  DBMS_OUTPUT.PUT_LINE('SIZE OF VARIABLE EXCEEDED');
      END;
      /
```

```
DATE
=========
16-SEP-21

PL/SQL procedure successfully completed.
```

****

QUERY-06: Write SQL code to create and execute an anonymous PL/SQL block that will check (say, for employee number 7108) whether an employee is entitled to receive the longevity bonus. Longevity bonus is given to employees with minimum 12 year of service. Now, re-execute the block to extend longevity bonus to employees with 10 years of service.

****

```
      DECLARE
          ID NUMBER(4) := &EMPLOYEE_ID;
          A EMPLOYEE%ROWTYPE;
      BEGIN
          SELECT * INTO A
          FROM EMPLOYEE
          WHERE ENO = ID;
          IF EXTRACT(YEAR FROM SYSDATE)-EXTRACT(YEAR FROM A.HIREDATE)>=12 THEN
```

```
            DBMS_OUTPUT.PUT_LINE('EMPLOYEE ELIGIBLE FOR LONGEVITY BONUS');
          ELSE
            DBMS_OUTPUT.PUT_LINE('EMPLOYEE NOT ELIGIBLE FOR LONGEVITY BONUS');
          END IF;
      END;
      /
```

```
Enter value for employee_id: 7111
old   2:        ID NUMBER(4) := &EMPLOYEE_ID;
new   2:        ID NUMBER(4) := 7111;
EMPLOYEE NOT ELIGIBLE FOR LONGEVITY BONUS
```

```
      DECLARE
          ID NUMBER(4) := &EMPLOYEE_ID;
          A EMPLOYEE%ROWTYPE;
      BEGIN
          SELECT * INTO A
          FROM EMPLOYEE
          WHERE ENO = ID;
          IF EXTRACT(YEAR FROM SYSDATE)-EXTRACT(YEAR FROM A.HIREDATE)>=10 THEN
            DBMS_OUTPUT.PUT_LINE('EMPLOYEE ELIGIBLE FOR LONGEVITY BONUS');
          ELSE
            DBMS_OUTPUT.PUT_LINE('EMPLOYEE NOT ELIGIBLE FOR LONGEVITY BONUS');
          END IF;
      END;
      /
```

```
Enter value for employee_id: 7110
old   2: ID NUMBER(4) := &EMPLOYEE_ID;
new   2: ID NUMBER(4) := 7110;
EMPLOYEE ELIGIBLE FOR LONGEVITY BONUS

PL/SQL procedure successfully completed.
```

****

QUERY-07: Write SQL code to create and execute an anonymous PL/SQL block that will locate the first August born employee. Re-write and execute an anonymous PL/SQL block that will locate the first August born employee, when EMPLOYEE table is searched in reversed order.

****

```
      DECLARE
            A EMPLOYEE%ROWTYPE;
      BEGIN
            SELECT *
            INTO A
            FROM EMPLOYEE
            WHERE UPPER(TO_CHAR(BIRTHDATE,'MON'))='AUG'
            AND ROWNUM=1;
            DBMS_OUTPUT.PUT_LINE(TO_CHAR(A.ENO)||' '||A.FNAME||' '||A.LNAME);
      END;
      /

7114 Larry Gomes

PL/SQL procedure successfully completed.
```

```
      DECLARE
            A EMPLOYEE%ROWTYPE;
      BEGIN
            SELECT *
            INTO A
            FROM EMPLOYEE
            WHERE UPPER(TO_CHAR(BIRTHDATE,'MON'))='AUG'
            AND ROWNUM=1
            ORDER BY ENO DESC;
            DBMS_OUTPUT.PUT_LINE(TO_CHAR(A.ENO)||' '||A.FNAME||' '||A.LNAME);
      END;
      /
7114 Larry Gomes

PL/SQL procedure successfully completed.
```

****

QUERY-08: Write SQL code to create and execute an anonymous PL/SQL block that accept staff ID from the console and will display staff details for said staff. A system exception, NO_DATA_FOUND should be cached when the mentioned staff does not exist.

/*

Create table PAYSCALE, that includes fields - DESIGNATION (15 alphanumeric characters), MINPAY (5 digits), MAXPAY (5 digits). Entity Integrity is maintained on DESIGNATION, with plausible values - Professor, Research Asst., Assa. Professor, Teaching Asst., and Asst. Professor.

Add following tuples to PAYSCALE table.

Professor, 140000,200000

Assa. Professor, 100000, 140000

Asst. Professor, 50000, 90000

Teaching Asst., 20000, 32500

Research Asst., 30000, 45000

***/**

****

```
	DECLARE
		ID NUMBER(3) := &STAFF_ID;
		A STAFF%ROWTYPE;
	BEGIN
		SELECT * INTO A FROM STAFF WHERE SID=ID;
		DBMS_OUTPUT.PUT_LINE('STAFF ID := '||A.SID);
		DBMS_OUTPUT.PUT_LINE('STAFF NAME := '||A.NAME);
		DBMS_OUTPUT.PUT_LINE('STAFF BRANCH := '||A.BRANCH);
		DBMS_OUTPUT.PUT_LINE('STAFF DESIGNATION := '||A.DESG);
		DBMS_OUTPUT.PUT_LINE('STAFF JOINING DATE := '||A.JOIN_DT);
	EXCEPTION
		WHEN NO_DATA_FOUND THEN
			DBMS_OUTPUT.PUT_LINE('NO DATA FOUND OF STAFF ID '||ID);
	END;
	/
```

```
Enter value for staff_id: 105
```

```
old   2:        ID NUMBER(3) := &STAFF_ID;
new   2:        ID NUMBER(3) := 105;
STAFF ID := 105
STAFF NAME := Geetika Goenka
STAFF BRANCH := CSEC
STAFF DESIGNATION := Professor
STAFF JOINING DATE := 15-NOV-09

PL/SQL procedure successfully completed.
```

```
      CREATE TABLE PAYSCALE (
            DESIGNATION VARCHAR2(15) NOT NULL,
            MINPAY NUMBER(8,2) NOT NULL,
            MAXPAY NUMBER(8,2) NOT NULL,
            CONSTRAINT PAYSCALE_PK_DESIGNATION PRIMARY KEY (DESIGNATION)
      );

Table created.
```

```
      INSERT INTO PAYSCALE
      VALUES ('Professor',140000.00,200000.00);

1 row created.

      INSERT INTO PAYSCALE
      VALUES ('Asso. Professor',100000.00,140000.00);

1 row created.

      INSERT INTO PAYSCALE
      VALUES ('Asst. Professor',50000.00,90000.00);

1 row created.

      INSERT INTO PAYSCALE
      VALUES ('Teaching Asst.',20000.00,32500.00);

1 row created.

      INSERT INTO PAYSCALE
      VALUES ('Research Asst.',30000.00,45000.00);

1 row created.
```

```
      SELECT *
      FROM PAYSCALE;

DESIGNATION         MINPAY     MAXPAY
-------------- ---------- ----------
Professor           140000     200000
Asso. Professor     100000     140000
Asst. Professor      50000      90000
Teaching Asst.       20000      32500
Research Asst.       30000      45000

5 rows selected.
```

****

QUERY-09: Write SQL code to create and execute an anonymous PL/SQL block that defines user-defined exceptions - BELOW_PAY_RANGE and ABOVE_PAY_RANGE. Your script should accept an employee number from the console and check for the salary to fall within the payscale [minpay, maxpay].

If the salary is less than minpay, BELOW_PAY_RANGE exception is raised and when cached an appropriate message -

'<EmpNo> Receives Salary Below Scale [minpay, maxpay]'

is displayed; otherwise ABOVE_PAY _RANGE exception is raised and cached to display the appropriate message accordingly.

You must appropriately catch the NO_DATA_FOUND exception also. When there are no violations, display for the employee the salary drawn. Test the above anonymous block for input employee numbers - 7101, 7104, 7106, 7109,7111, 7114 and 7117.

****

```
DECLARE
      EMPNO NUMBER(4) := &EMPLOYEE_NUMBER;
      A EMPLOYEE%ROWTYPE;
      BELOW_PAY_RANGE EXCEPTION;
      ABOVE_PAY_RANGE EXCEPTION;
      B PAYSCALE%ROWTYPE;
BEGIN
      SELECT * INTO A FROM EMPLOYEE WHERE ENO=EMPNO;
      SELECT * INTO B FROM PAYSCALE WHERE UPPER(DESIGNATION)=UPPER(A.DESIGNATION);
      IF A.SALARY>=B.MAXPAY THEN
            RAISE ABOVE_PAY_RANGE;
      ELSIF A.SALARY<=B.MINPAY THEN
            RAISE BELOW_PAY_RANGE;
      ELSE
            DBMS_OUTPUT.PUT_LINE(TO_CHAR(EMPNO)||' RECEIVES SALARY '
                                                ||TO_CHAR(A.SALARY));

      END IF;
EXCEPTION
      WHEN BELOW_PAY_RANGE THEN
            DBMS_OUTPUT.PUT_LINE(TO_CHAR(EMPNO)||' RECEIVES SALARY BELOW PAY RANGE
                                    '||TO_CHAR(B.MINPAY)||' AND '||TO_CHAR(B.MAXPAY));
      WHEN ABOVE_PAY_RANGE THEN
            DBMS_OUTPUT.PUT_LINE(TO_CHAR(EMPNO)||' RECEIVES SALARY ABOVE PAY RANGE
                                    '||TO_CHAR(B.MINPAY)||' AND '||TO_CHAR(B.MAXPAY));
      WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE(TO_CHAR(EMPNO)||' DATA NOT FOUND IN TABLE');
 END;
 /

Enter value for employee_number: 7101
old   2: EMPNO NUMBER(4) := &EMPLOYEE_NUMBER;
new   2: EMPNO NUMBER(4) := 7101;
7101 RECEIVES SALARY 150000

PL/SQL procedure successfully completed.


/
Enter value for employee_number: 7104
old   2: EMPNO NUMBER(4) := &EMPLOYEE_NUMBER;
new   2: EMPNO NUMBER(4) := 7104;
7104 RECEIVES SALARY BELOW PAY RANGE 140000 AND 200000

PL/SQL procedure successfully completed.
```

```
/
Enter value for employee_number: 7106
old   2: EMPNO NUMBER(4) := &EMPLOYEE_NUMBER;
new   2: EMPNO NUMBER(4) := 7106;
7106 RECEIVES SALARY 127400

PL/SQL procedure successfully completed.


/
Enter value for employee_number: 7109
old   2: EMPNO NUMBER(4) := &EMPLOYEE_NUMBER;
new   2: EMPNO NUMBER(4) := 7109;
7109 RECEIVES SALARY ABOVE PAY RANGE 50000 AND 90000

PL/SQL procedure successfully completed.


/
Enter value for employee_number: 7111
old   2: EMPNO NUMBER(4) := &EMPLOYEE_NUMBER;
new   2: EMPNO NUMBER(4) := 7111;
7111 RECEIVES SALARY ABOVE PAY RANGE 30000 AND 45000

PL/SQL procedure successfully completed.


/
Enter value for employee_number: 7114
old   2: EMPNO NUMBER(4) := &EMPLOYEE_NUMBER;
new   2: EMPNO NUMBER(4) := 7114;
7114 RECEIVES SALARY ABOVE PAY RANGE 20000 AND 32500

PL/SQL procedure successfully completed.


/
Enter value for employee_number: 7117
old   2: EMPNO NUMBER(4) := &EMPLOYEE_NUMBER;
new   2: EMPNO NUMBER(4) := 7117;
7117 RECEIVES SALARY 32200
```

```
PL/SQL procedure successfully completed.
```

****

QUERY-10: Write a SQL code to create and execute an anonymous PL/SQL block that will modify Query-09 to process all records of EMPLOYEE table. You need not acquire employee number from console. You should only report the violations.

****

```
      DECLARE
            A EMPLOYEE%ROWTYPE;
            B PAYSCALE%ROWTYPE;
            BELOW_PAY_RANGE EXCEPTION;
            ABOVE_PAY_RANGE EXCEPTION;
            I NUMBER(4);
             BEGIN
            FOR I IN 7101..7117 LOOP
                  BEGIN
                        SAVEPOINT S;
                        SELECT * INTO A
                        FROM EMPLOYEE
                        WHERE ENO=I;
                        SELECT * INTO B
                        FROM PAYSCALE
                        WHERE UPPER(DESIGNATION)=UPPER(A.DESIGNATION);
                        IF A.SALARY>=B.MAXPAY THEN
                              RAISE ABOVE_PAY_RANGE;
                        ELSIF A.SALARY<=B.MINPAY THEN
                              RAISE BELOW_PAY_RANGE;
                        ELSE
                              DBMS_OUTPUT.PUT_LINE(TO_CHAR(I)||' RECEIVES SALARY
'
                                                                  ||TO_CHAR(A.SALAR
                                                           Y));
                        END IF;
                  EXCEPTION
                        WHEN BELOW_PAY_RANGE THEN
                              DBMS_OUTPUT.PUT_LINE(TO_CHAR(I)||
                                           ' RECEIVES SALARY BELOW PAY RANGE’||
                                           TO_CHAR(B.MINPAY)||' AND '||
                                           TO_CHAR(B.MAXPAY));
                              ROLLBACK TO SAVEPOINT S;
                        WHEN ABOVE_PAY_RANGE THEN
                              DBMS_OUTPUT.PUT_LINE(TO_CHAR(I)||
                                           ' RECEIVES SALARY ABOVE PAY RANGE '||
                                           TO_CHAR(B.MINPAY)||' AND '||
                                           TO_CHAR(B.MAXPAY));
                              ROLLBACK TO SAVEPOINT S;
                          WHEN NO_DATA_FOUND THEN
                              DBMS_OUTPUT.PUT_LINE(TO_CHAR(I)||
                                           ' DATA NOT FOUND IN TABLE');
                              ROLLBACK TO SAVEPOINT S;
                  END;
            END  LOOP;
      END;
      /
```

```
7101 RECEIVES SALARY 150000
7102 RECEIVES SALARY 146500
7103 RECEIVES SALARY 148000
7104 RECEIVES SALARY BELOW PAY RANGE 140000 AND 200000
7105 RECEIVES SALARY 127400
7106 RECEIVES SALARY 127400
7107 RECEIVES SALARY 127400
7108 RECEIVES SALARY 119700
7109 RECEIVES SALARY ABOVE PAY RANGE 50000 AND 90000
7110 RECEIVES SALARY 86400
7111 RECEIVES SALARY ABOVE PAY RANGE 30000 AND 45000
7112 RECEIVES SALARY 44600
7113 RECEIVES SALARY ABOVE PAY RANGE 20000 AND 32500
7114 RECEIVES SALARY ABOVE PAY RANGE 20000 AND 32500
7115 RECEIVES SALARY 30000
```

```
7116 RECEIVES SALARY 30000
7117 RECEIVES SALARY 32200

PL/SQL procedure successfully completed.
```

****

QUERY-11: Write a SQL code to compile and execute an anonymous block which declares a cursor - FACULTY. The cursor buffers the records comprising - Employee ID, Employee Name (FNAME and LNAME combined) and Designation for the Designation entered by the user.

You may use either EMPLOYEE table or EMPP table for this cursor and print the buffered records. Use %NOTFOUND variable to enable cursor exit.

****

```
DECLARE
      CURSOR FACULTY IS
            SELECT *
            FROM EMPP
            WHERE UPPER(DESIGNATION) LIKE UPPER('&DESIGNATION');
      A EMPP%ROWTYPE;
      I INT := 1;
BEGIN
      OPEN FACULTY;
      LOOP
            FETCH FACULTY INTO A;
            EXIT WHEN FACULTY%NOTFOUND;
            DBMS_OUTPUT.PUT_LINE(RPAD(TO_CHAR(A.EID),11)
                                ||'  '||RPAD(A.ENAME,21)
                                ||'  '||RPAD(A.DESIGNATION,15));
            I := 0;
      END LOOP;
IF I=0 THEN
      DBMS_OUTPUT.PUT_LINE('ALL CURSOR ROWS FETCHED ...');
ELSE
      DBMS_OUTPUT.PUT_LINE('NO MATCHING ROWS FETCHED ...');
END IF;
CLOSE FACULTY;
END;
/
```

```
Enter value for designation: LAMBDA
```

```
old   5:  WHERE UPPER(DESIGNATION) LIKE UPPER('&DESIGNATION');
new   5:  WHERE UPPER(DESIGNATION) LIKE UPPER('LAMBDA');
NO MATCHING ROWS FETCHED ...

PL/SQL procedure successfully completed.
```

```
      /

Enter value for designation: ProFessOr
old   5:  WHERE UPPER(DESIGNATION) LIKE UPPER('&DESIGNATION');
new   5:  WHERE UPPER(DESIGNATION) LIKE UPPER('ProFessOr');
7101         Eugene Sabatini        Professor
7102         Samantha Jones         Professor
7103         Alexander Lloyd        Professor
7104         Simon Downing          Professor
ALL CURSOR ROWS FETCHED ...

PL/SQL procedure successfully completed.
```

****

QUERY-12: CURSOR FOR LOOP:

Modify the cursor in Query-01 as FACULTY_CFL which uses the cursor FOR loop to buffering and displaying the records (as mentioned) when employee designation is entered by the user.

Use a variation of cursor FOR loop to include the ROWCOUNT variable sto print serial number for the displayed records.

****

```
DECLARE
      CURSOR FACULTY_CFL IS
                  SELECT *
                  FROM EMPP
                  WHERE UPPER(DESIGNATION) LIKE UPPER('&DESIGNATION');
BEGIN
      DBMS_OUTPUT.PUT_LINE('THE CURSOR FOR LOOP ...');
      FOR A IN FACULTY_CFL LOOP
            DBMS_OUTPUT.PUT_LINE(RPAD(TO_CHAR(A.EID),11)||
                                 '  '||RPAD(A.ENAME,21)||'  '||
                                 RPAD(A.DESIGNATION,15));
```

```
END LOOP;
DBMS_OUTPUT.PUT_LINE(CHR(10));
DBMS_OUTPUT.PUT_LINE('THE CURSOR FOR LOOP WITH %ROWCOUNT ...');
FOR A IN FACULTY_CFL LOOP
DBMS_OUTPUT.PUT_LINE(RPAD(FACULTY_CFL%ROWCOUNT,4)
                          ||'  '||RPAD(TO_CHAR(A.EID),11)
                          ||'  '||RPAD(A.ENAME,21)||
                          '  '||RPAD(A.DESIGNATION,15));
END LOOP;
DBMS_OUTPUT.PUT_LINE('CURSOR FOR LOOP EXITED ...');
END;
/
```

```
Enter value for designation: proFEssor
old   5:  WHERE UPPER(DESIGNATION) LIKE UPPER('&DESIGNATION');
new   5:  WHERE UPPER(DESIGNATION) LIKE UPPER('proFEssor');
THE CURSOR FOR LOOP ...
7101         Eugene Sabatini        Professor
7102         Samantha Jones         Professor
7103         Alexander Lloyd        Professor
7104         Simon Downing          Professor


THE CURSOR FOR LOOP WITH %ROWCOUNT ...
1     7101         Eugene Sabatini        Professor
2     7102         Samantha Jones         Professor
3     7103         Alexander Lloyd        Professor
4     7104         Simon Downing          Professor
CURSOR FOR LOOP EXITED ...

PL/SQL procedure successfully completed.
```

****

QUERY-13: EXITING A CURSOR AFTER FETCHING SPECIFIED NUMBER OF ROWS:
Modify the cursor FACULTY_CFL_A to display only those many records as desired by the user. Use %ROWCOUNT to enable the cursor to ensure this.
****

```
 DECLARE
       CURSOR FACULTY_CFL_A IS
                    SELECT *
                    FROM EMPP
                    WHERE UPPER(DESIGNATION) LIKE UPPER('&DESIGNATION');
       A FACULTY_CFL_A%ROWTYPE;
       R NUMBER := &HOW_MANY_ROWS;
BEGIN
       OPEN FACULTY_CFL_A;
       LOOP
              FETCH FACULTY_CFL_A INTO A;
              EXIT WHEN FACULTY_CFL_A%ROWCOUNT > R OR FACULTY_CFL_A%NOTFOUND;
              DBMS_OUTPUT.PUT_LINE(RPAD(FACULTY_CFL_A%ROWCOUNT,4)
                                    ||'  '||RPAD(TO_CHAR(A.EID),11)
                                    ||'  '||RPAD(A.ENAME,21)
                                    ||'  '||RPAD(A.DESIGNATION,15));
       END LOOP;
       CLOSE FACULTY_CFL_A;
END;
/
```

```
Enter value for designation: professor
old   5:  WHERE UPPER(DESIGNATION) LIKE UPPER('&DESIGNATION');
new   5:  WHERE UPPER(DESIGNATION) LIKE UPPER('professor');
Enter value for how_many_rows: 3
old   7:  R NUMBER := &HOW_MANY_ROWS;
new   7:  R NUMBER := 3;
1     7101         Eugene Sabatini        Professor
2     7102         Samantha Jones         Professor
3     7103         Alexander Lloyd        Professor

PL/SQL procedure successfully completed.
```

```
 /
```

```
Enter value for designation: PRoFeSSor
old   5:  WHERE UPPER(DESIGNATION) LIKE UPPER('&DESIGNATION');
new   5:  WHERE UPPER(DESIGNATION) LIKE UPPER('PRoFeSSor');
Enter value for how_many_rows: 5
old   7:  R NUMBER := &HOW_MANY_ROWS;
new   7:  R NUMBER := 5;
1     7101         Eugene Sabatini        Professor
```

```
2     7102         Samantha Jones         Professor
3     7103         Alexander Lloyd        Professor
4     7104         Simon Downing          Professor

PL/SQL procedure successfully completed.
```

****

PARAMETERIZED CURSOR WITH DEFAULT VALUES:

Write a SQL code to compile and execute an anonymous block which declares a cursor - EMP_SAL_INFO (Salary, Designation). Let the default values for salary and designation be 75000 and " Asst. Professor" respectively.

The cursor buffers the records comprising - Employee ID, Employee Name (FNAME and LNAME combined), Designation and Salary for the Salary and Designation entered by the user. Use EMPLOYEE table for this cursor. Use this cursor to print the buffered records.

****

```
 DECLARE
       CURSOR EMP_SAL_INFO (S EMPLOYEE.SALARY%TYPE
                                     DEFAULT 75000,
                            D EMPLOYEE.DESIGNATION%TYPE
                                     DEFAULT 'Asst. Professor') IS
             SELECT ENO AS EMPLOYEE_ID,
                           CONCAT(CONCAT(FNAME,' '),LNAME)
                           AS EMPLOYEE_NAME,
                           DESIGNATION,SALARY
             FROM EMPLOYEE
             WHERE SALARY > S AND UPPER(DESIGNATION) = UPPER(D);
      SAL EMPLOYEE.SALARY%TYPE;
      DES EMPLOYEE.DESIGNATION%TYPE;
BEGIN
       DBMS_OUTPUT.PUT_LINE('WITH DEFAULT VALUES ...');
       FOR B IN EMP_SAL_INFO LOOP
       DBMS_OUTPUT.PUT_LINE(RPAD(B.EMPLOYEE_ID,11)
                            ||'  '||RPAD(B.EMPLOYEE_NAME,21)
                            ||'  '||RPAD(B.DESIGNATION,15)
                            ||' '||RPAD(B.SALARY,8));
```

```
            END LOOP;
            DBMS_OUTPUT.PUT_LINE(CHR(10));
            SAL := '&SPECIFIED_SALARY';
            DBMS_OUTPUT.PUT_LINE('WITH SOME DEFAULT VALUES ...');
            FOR B IN EMP_SAL_INFO(SAL) LOOP
                  DBMS_OUTPUT.PUT_LINE(RPAD(B.EMPLOYEE_ID,11)
                                    ||'  '||RPAD(B.EMPLOYEE_NAME,21)
                                    ||'  '||RPAD(B.DESIGNATION,15)
                                    ||' '||RPAD(B.SALARY,8));
            END LOOP;
            DBMS_OUTPUT.PUT_LINE(CHR(10));
            SAL := '&SPECIFIED_SALARY';
            DES := '&SPECIFIED_DESIGNATION';
            DBMS_OUTPUT.PUT_LINE('WITH ALL SUPPLIED VALUES ...');
            FOR B IN EMP_SAL_INFO(SAL,DES) LOOP
                  DBMS_OUTPUT.PUT_LINE(RPAD(B.EMPLOYEE_ID,11)
                                    ||'  '||RPAD(B.EMPLOYEE_NAME,21)
                                    ||'  '||RPAD(B.DESIGNATION,15)
                                    ||' '||RPAD(B.SALARY,8));
            END LOOP;
            DBMS_OUTPUT.PUT_LINE(CHR(10));
      DBMS_OUTPUT.PUT_LINE('ALL CASES DONE ...');
      END;
      /
```

```
Enter value for specified_salary: 88000
old  14: SAL := '&SPECIFIED_SALARY';
new  14: SAL := '88000';
Enter value for specified_salary: 120000
old  20: SAL := '&SPECIFIED_SALARY';
new  20: SAL := '120000';
Enter value for specified_designation: Asso. Professor
old  21: DES := '&SPECIFIED_DESIGNATION';
new  21: DES := 'Asso. Professor';
WITH DEFAULT VALUES ...
7109         Martina Jacobson       Asst. Professor 91000
7110         William Smithfield     Asst. Professor 86400


WITH SOME DEFAULT VALUES ...
7109         Martina Jacobson       Asst. Professor 91000
```

```
WITH ALL SUPPLIED VALUES ...
7107        Christov Plutnik       Asso. Professor 127400
7105        Christina Mulboro      Asso. Professor 127400
7106        Dolly Silverline       Asso. Professor 127400


ALL CASES DONE ...
```

****

QUERY-15: BULK COLLECT with CURSORS:

Write SQL code to compile and execute a procedure - PRINT_EMPLOYEE which receives employee salary as input and prints the following particulars - employee number, employee name and salary, for employees whose salary exceeds the inputted salary.
You must use a cursor - SAL_CURSOR, to buffer required result-set for bulk collect. Use TYPE statement to declare and instantiate array variables.
You may also try using %ROWCOUNT. Use EMPP table as source. You may also use EMPLOYEE table.

****

```
CREATE OR REPLACE PROCEDURE PRINT_EMPLOYEE (S EMPP.SALARY%TYPE)
      AS TYPE NUM_ARRAY IS VARRAY(10000) OF NUMBER(4);
      TYPE STR_ARRAY IS VARRAY(10000) OF VARCHAR(21);
      TYPE N_ARRAY   IS VARRAY(10000) OF NUMBER(8,2);
      EMPLOYEE_ID_ARRAY NUM_ARRAY;
      EMPLOYEE_NAME_ARRAY STR_ARRAY;
      EMPLOYEE_SALARY_ARRAY N_ARRAY;
      CURSOR SAL_CURSOR IS
                  SELECT EID, ENAME, SALARY
                  FROM EMPP
                  WHERE SALARY > S;
BEGIN
      OPEN SAL_CURSOR;
      FETCH SAL_CURSOR
      BULK COLLECT INTO EMPLOYEE_ID_ARRAY,
                        EMPLOYEE_NAME_ARRAY,EMPLOYEE_SALARY_ARRAY;
      CLOSE SAL_CURSOR;
      DBMS_OUTPUT.PUT_LINE('EMPLOYEE HAVING SALARY > '||S);
```

```
            DBMS_OUTPUT.PUT_LINE('  EID          EMPLOYEE_NAME    SALARY');
            DBMS_OUTPUT.PUT_LINE('-----  ---------------------  --------');
            FOR I IN EMPLOYEE_ID_ARRAY.FIRST .. EMPLOYEE_ID_ARRAY.LAST LOOP
            DBMS_OUTPUT.PUT_LINE(RPAD(EMPLOYEE_ID_ARRAY(I),5)
                                ||' '||RPAD(EMPLOYEE_NAME_ARRAY(I),21)
                                ||' '||LPAD(EMPLOYEE_SALARY_ARRAY(I),10));
            END LOOP;
            DBMS_OUTPUT.PUT_LINE('-----  ---------------------  --------');
            DBMS_OUTPUT.PUT_LINE('... END OF BULK FETCH ...');
      END;
      /

Procedure created.

CALL PRINT_EMPLOYEE(50000);

EMPLOYEE HAVING SALARY > 50000
EID           EMPLOYEE_NAME     SALARY
----- --------------------   --------
7101   Eugene Sabatini          150000
7102   Samantha Jones           146500
7103   Alexander Lloyd          148000
7104   Simon Downing            138400
7105   Christina Mulboro        127400
7106   Dolly Silverline         127400
7107   Christov Plutnik         127400
7108   Ellena Sanchez           119700
7109   Martina Jacobson          91000
7110   William Smithfield        86400
-----  --------------------  --------
... END OF BULK FETCH ...

Call completed.

CALL PRINT_EMPLOYEE(125000);

EMPLOYEE HAVING SALARY > 125000
EID           EMPLOYEE_NAME     SALARY
-----  --------------------  --------
7101   Eugene Sabatini          150000
7102   Samantha Jones           146500
7103   Alexander Lloyd          148000
```

```
7104   Simon Downing           138400
7105   Christina Mulboro       127400
7106   Dolly Silverline        127400
7107   Christov Plutnik        127400
-----  --------------------  --------
... END OF BULK FETCH ...

Call completed.
```

CALL PRINT_EMPLOYEE(148000);

```
EMPLOYEE HAVING SALARY > 148000
EID            EMPLOYEE_NAME    SALARY
-----  --------------------  --------
7101   Eugene Sabatini         150000
-----  --------------------  --------
... END OF BULK FETCH ...

Call completed.
```

Conclusion:

In this experiment, we learnt about the PL/SQL blocks, exceptions and cursors. PL/SQL, the Oracle procedural extension of SQL, is a portable, high-performance transaction-processing language. This chapter outlines its benefits and briefly describes its key features and architecture. Errors are straightforward to discover and handle in PL/SQL. PL/SQL throws an exception when an error occurs. Normal execution is halted, and control is transferred to the PL/SQL block's exception-handling section. As with a C program, you do not have to inspect every operation to confirm that it was successful. The block, which unites related declarations and statements, is the basic unit of a PL/SQL source programme. The terms DECLARE, BEGIN, EXCEPTION, and END define a PL/SQL block. These keywords split the block into three sections: declarative, executable, and exception-handling. Only the executable portion is needed. A label can be applied to a block. Because the block is not recorded in the database, it is referred to as an anonymous block (even if it has a label).Every time an anonymous block is loaded into memory, it is compiled in three stages: The syntax of PL/SQL is verified, and a parse tree is created. Semantic checking-Type checking and further parse tree processing, Creating Code. A cursor is a pointer to a private SQL region where information about processing a single SQL query or PL/SQL SELECT INTO command is stored. The cursor may be used to fetch the rows of the result set one at a time. Cursor attributes may be used to obtain information about the cursor's current status, such as how many rows the statement has touched thus far. The %ROWTYPE property allows you to specify whether a record represents a complete or partial row of a database table or view. The record has a field with the same name and data type for each column of the entire or partial row. If the structure of the row changes, so does the structure of the record. The %TYPE property allows you to declare a data item of the same data type as an already specified variable or column (without knowing what that type is). If the declaration of the referenced item changes, so does the declaration of the referring item. When defining variables to contain database values, the %TYPE property comes in handy.

Viva Questions:

1. **What is an anonymous block?**

   The anonymous block statement in PL/SQL is an executable statement that can include PL/SQL control statements and SQL statements. It may be used in a scripting language to construct procedural logic. This statement can be generated and executed by the data server in PL/SQL situations. The anonymous block statement, which does not persist in the database, can include up to three sections: a declaration part that is optional, an obligatory executable section, and an optional exception section. The optional declaration section is put before the executable BEGIN-END block and can contain the declaration of variables, cursors, and types that will be utilised by statements within the executable and exception sections. The exception section is optional and can be included at the conclusion of the BEGIN-END block. The exception section must start with the keyword EXCEPTION and run to the end of the block in which it appears.

2. **What is an exception? List the standard PL/SQL exceptions.**

   An exception is an incorrect circumstance that occurs during the execution of a programme. PL/SQL allows programmers to catch such errors by utilising the EXCEPTION block in the programme, and then take necessary action against the error situation. Exceptions are classified into two types:
   - System-defined exceptions
   - User-defined exceptions

   The following are the 21 standard PL/SQL exceptions:

- ACCESS_INTO_NULL
- CASE_NOT_FOUND
- COLLECTION_IS_NULL
- DUP_VAL_ON_INDEX
- INVALID_CURSOR
- INVALID_NUMBER
- LOGIN_DENIED
- NO_DATA_FOUND
- NOT_LOGGED_ON
- PROGRAM_ERROR
- ROWTYPE_MISMATCH
- SELF_IS_NULL
- STORAGE_ERROR
- TOO_MANY_ROWS
- VALUE_ERROR

- ZERO_DIVIDE
- CURSOR_ALREADY_OPEN
- SUBSCRIPT_BEYOND_COUNT
- SUBSCRIPT_BEYOND_LIMIT
- SYS_INVALID_ROUND
- TIMEOUT_ON_RESOURCE

3. Differentiate between '&' and '&&' in SQL.

The character "&" is used to establish a temporary replacement variable. Every time the variable is mentioned, you will be requested to input the value.

The character “&&” is used to define a variable for permanent replacement. You just need to input the value once.

4. Why it is a good practice to use %TYPE when declaring variables?

The data server supports the %TYPE property, which is used in PL/SQL variable and parameter declarations. The use of this feature guarantees the type compatibility of table columns and PL/SQL variables. As a prefix to the %TYPE attribute, a qualified column name in dot notation or the name of a previously defined variable must be given. The variable being defined is given the data type of this column or variable. There is no need to alter the declaration code if the data type of the column or variable changes.

5. What is a cursor? List the steps associated with implementing a cursor.

The Oracle engine uses a work area to analyse and store information internally while executing SQL queries. This work area is exclusive to SQL operations. The PL/SQL construct 'Cursor' allows the user to name the work area and retrieve the information contained in it. A cursor's primary role is to obtain data from a result set one row at a time, as opposed to SQL commands, which work on all rows in the result set at once. Cursors are utilised when a user needs to change records in a database table in a singleton or row by row basis.

Using a Cursor involves four stages.

- IN THE DECLARATION SECTION, DECLARE THE CURRENT.
- OPEN the Execution Section using your pointer.
- In the Execution Section, FETCH the cursor's data into PL/SQL variables or records.
- Before you conclude the PL/SQL Block, CLOSE the cursor in the Execution Section.

6. What is an" active set"?

The data kept in the Cursor is referred to as the Active Data Set. Oracle DBMS features a second designated region in the main memory Set where cursors are opened. As a result, the size of the cursor is constrained by the size of this pre-defined region.

7. What are the advantages of a cursor FOR loop?

The PL/SQL cursor FOR loop provides the benefit of continuing the loop until the row is not found. You may need to use an explicit cursor with a FOR loop instead of the OPEN, FETCH, and CLOSE statements at times. The FOR loop iterates repeatedly, retrieving rows of values from the database until the row is not found.

8. **Why it is a good practice to close a cursor?**

When you open a cursor, Oracle executes the query to create the results and places the cursor before the first row of the result set. A cursor, on the other hand, may only be opened if it is not already open; attempting to open a cursor that is already open results in a "CURSOR ALREADY OPEN" exception. In other words, if you define a cursor and open it, Oracle throws an exception if you try to open it again without closing it. If you finish with the cursor but don't close it, nothing occurs unless, as previously said, you try to open it or fetch from it again (assuming all the data has been read). However, there are a few of hazards to be aware of. If you have a significant number of open cursors, you may surpass the Oracle database initialization option OPEN CURSORS, which is the limit for the maximum number of open cursors per session, or your database may run out of memory. Either of these may certainly pose severe issues for an application. In other words, closing your cursors in your PL/SQL scripts after you're done with them is excellent practise. Nothing will happen if you do not. In other words, closing your cursors in your PL/SQL scripts once you've finished with them is excellent practise. If you don't, nothing will happen for a time, but your Oracle database applications may cease operating unexpectedly or slowly.