

**S.Y.B.Sc**

**Computer Science**

**2019 – 2020**

**This is to certify that PRASHANT SINGH of S.Y.B.Sc Roll No. 70 has successfully completed the practical of Paper – IV (sem – III)** **Database Management System during the Academic Year 2019-2020 as specified by the MUMBAI UNIVERSITY.**

**Prof. (Mrs.) Anita Gaikwad Prof. (Mrs.) AnitaGaikwad**

**Professor-In-charge In-charge**

**Dept. of Computer Science Dept. of Computer Science**

**Name: PRASHANT SINGH Subject: DBMS**

**Class: S.Y.BSC (COMPUTER SCIENCE) Roll No: 070**

**Paper I Section I**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **INDEX** | | | | |
| **NO** | **DATE** | **TITLE** | **PAGE NO** | **SIGN** |
|  | | | | |
| **1** |  | Creating and working with Insert/Update/Delete Trigger using Before/After clause. |  |  |
|  | | | | |
| **2** |  | Writing PL/SQL Blocks with basic programming constructs by including following:  a. Sequential Statements  b. unconstrained loop |  |  |
|  | | | | |
| **3** |  | Sequences  Sequences:  Sequences:  Sequences:  Sequences:  Sequences: |  |  |
|  | | | | |
| **4** |  | Writing PL/SQL Blocks with basic programming constructs by including following:  If...then...Else,  IF...ELSIF...ELSE... END IF  b. Case statement |  |  |
|  | | | | |
| **5** |  | Writing PL/SQL Blocks with basic programming constructs for following Iterative Structure:  a. While-loop Statements  b. For-loop Statements |  |  |
|  | | | | |
| **6** |  | Writing PL/SQL Blocks with basic programming constructs by including a Go TO jump out of a loop and NULL as a statement inside IF |  |  |
|  | | | | |
| **7** |  | Writing Procedures in PL/SQL Block |  |  |
|  | | | | |
| **8** |  | Writing Functions in PL/SQL Block. |  |  |
|  | | | | |
| **9** |  | Study of transactions and locks |  |  |
|  | | | | |
| **10** |  | Extra Examples of PLSQL |  |  |
|  |  |  |  |  |
| **11** |  | Creating a Cursors in Pl/sql functions |  |  |
|  |  |  |  |  |
| **12** |  | Creating packages in PL/SQL |  |  |

1. **Creating and working with Insert/Update/Delete Trigger using Before/After clause.**

To start with, we will be using the CUSTOMERS table −

Select \* from customers;

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

+----+----------+-----+-----------+----------+

The following program creates a **row-level** trigger for the customers table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old values and new values −

CREATE OR REPLACE TRIGGER display\_salary\_changes

BEFORE DELETE OR INSERT OR UPDATE ON customers

FOR EACH ROW

WHEN (NEW.ID > 0)

DECLARE

sal\_diff number;

BEGIN

sal\_diff := :NEW.salary - :OLD.salary;

dbms\_output.put\_line('Old salary: ' || :OLD.salary);

dbms\_output.put\_line('New salary: ' || :NEW.salary);

dbms\_output.put\_line('Salary difference: ' || sal\_diff);

END;

/

When the above code is executed at the SQL prompt, it produces the following result −

Trigger created.

Let us perform some DML operations on the CUSTOMERS table. Here is one INSERT statement, which will create a new record in the table −

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)

VALUES (7, 'Kriti', 22, 'HP', 7500.00 );

When a record is created in the CUSTOMERS table, the above create trigger, **display\_salary\_changes** will be fired and it will display the following result −

Old salary:

New salary: 7500

Salary difference:

Because this is a new record, old salary is not available and the above result comes as null. Let us now perform one more DML operation on the CUSTOMERS table. The UPDATE statement will update an existing record in the table −

UPDATE customers

SET salary = salary + 500

WHERE id = 2;

When a record is updated in the CUSTOMERS table, the above create trigger, **display\_salary\_changes** will be fired and it will display the following result −

Old salary: 1500

New salary: 2000

Salary difference: 500

1. **Writing PL/SQL Blocks with basic programming constructs by including following:**
2. **Sequential Statements b. unconstrained loop**

**Q 1. Assuming sales table consisting of column zone, prodid and quantity. Write a PL/SQL code to accept zone and product id from the user to display total sale of specified product and zone with appropriate labels.**

ZONE PRODI QUANTITY

----- ----- ----------

aaa 101 300

bbb 102 400

bbb 102 550

bbb 102 450

ccc 103 200

ccc 103 6600

ddd 104 600

ddd 104 650

SET serveroutput ON;

DECLARE

v\_zone sales.zone%TYPE:=’&Zone’;

v\_prodid sales.prodid%TYPE:=’&Product\_Id’;

v\_quantity sales.quantity%TYPE;

v\_TotalSales sales.quantity%TYPE;

BEGIN

SELECT SUM(quantity)

INTO v\_TotalSales

FROM Sales

WHERE prodid=v\_prodid

AND zone=v\_zone;

DBMS\_OUTPUT.PUT\_LINE(‘#RESULT#’);

DBMS\_OUTPUT.PUT\_LINE(‘Product\_Id: ’||v\_prodid);

DBMS\_OUTPUT.PUT\_LINE(‘Zone ’||v\_zone);

DBMS\_OUTPUT.PUT\_LINE(‘Total Sales: ’||NVL(v\_TotalSales,0));

End;

o/p:

Enter value for zone: ccc

old 2: v\_zone sales.zone%TYPE:='&Zone';

new 2: v\_zone sales.zone%TYPE:='ccc';

Enter value for product\_id: 103

old 3: v\_prodid sales.prodid%TYPE:='&Product\_Id';

new 3: v\_prodid sales.prodid%TYPE:='103';

#RESULT#

Product\_Id: 103

Zone ccc

Total Sales: 6800

**Q.2 Write PL/SQL block to remove all rows of specified zone in above sales table.**

DECLARE

V\_zone sales.zone%type:='&Zone';

BEGIN

DELETE

From Sales

Where zone=v\_zone;

DBMS\_OUTPUT.PUT\_LINE('#RESULT#');

DBMS\_OUTPUT.PUT\_LINE('Zone'||v\_zone||' is deleted');

END;

SQL> /

Enter value for zone: ccc

old 2: V\_zone sales.zone%type:='&Zone';

new 2: V\_zone sales.zone%type:='ccc';

#RESULT#

Zoneccc is deleted

PL/SQL procedure successfully completed.

**Q 3. PLSQL block to calculate area and circumference of circle**

Declare

r number;

c number;

a number;

begin

r:=&r;

a:=3.14\*r\*r;

c:=2\*3.14\*r;

dbms\_output.put\_line(‘area of circle ‘||a);

dbms\_output.put\_line(‘cicumference of circle ‘||c);

end;

1. **Sequences:**
2. **Creating simple Sequences with clauses like START WITH, INCREMENT BY, MAXVALUE, MINVALUE, CYCLE | NOCYCLE, CACHE | NOCACHE, ORDER | NOORECER.**
3. **Creating and using Sequences for tables.**

Sequence is a user created database object that can be shared by multiple users to generate unique integers.

Syntax:

CREATE SEQUENCE\_ name

INCREMENT BY value

START WITH value

MINVALUE value

MAXVALUE value

CYCLE | NOCYCLE

CACHE value | NOCACHE;

1. Create sequence named STUD\_STUDID\_SEQ to be used for primary key of the STUDENT table. The sequence must start with 100 and every key value should be multiple of 5 it must not exceed 9999 and a key value must not be repeated.

CREATE SEQUENCE stud\_studid\_seq

INCREMENT BY 5

START WITH 100

MAXVALUE 9999

NOCACHE

NOCYCLE;

Sequence created.

1. Create a sequence named STUD\_ROLLNO\_SEQ to be used for the allotting roll numbers to STUDENT table.

CREATE SEQUENCE stud\_Rollno\_seq

INCREMENT BY 1

START WITH 1

MAXVALUE 80

NOCACHE

NOCYCLE;

Sequence created.

**Referencing Sequence**

1. **NEXTVAL**

Insert into students(stud\_id,student\_name,class\_id) values (STUD\_ROLLNO\_SEQ.NEXTVAL,’SUHAS’,25);

1. **CURRVAL**

Select dept\_deptid\_seq.CURRVAL from dual;

1. **Writing PL/SQL Blocks with basic programming constructs by including following:**
2. **If...then...Else, IF...ELSIF...ELSE... END IF**
3. **Case statement**
4. Write a PLSQL block to determine the salary increment of given employee based on the hire date of the employee.

create table employees

(

emp\_id number(3),

e\_name varchar2(10),

hire\_date date,

salary number(5));

select \* from employees;

EMP\_ID E\_NAME HIRE\_DATE SALARY

------ ---------- --------- ----------

1 manish 12-FEB-08 5000

2 ritesh 07-MAR-88 8000

3 rakesh 07-MAR-99 9000

4 pritam 27-MAY-93 20000

5 priteshm 02-JUL-03 7000

SQL> Declare

2 v\_bonus number(6,2);

3 v\_emp\_id number(6):=&Employee\_ID;

4 v\_hire\_date DATE;

5 BEGIN

6 Select hire\_date

7 INTO v\_hire\_date

8 FROM employees

9 WHERE emp\_id=v\_emp\_id;

10 IF v\_hire\_date > TO\_DATE('01-jan-98') THEN

11 V\_bonus:=500;

12 ELSIF v\_hire\_date > TO\_DATE('01-JAN-96') THEN

13 V\_bonus:=1000;

14 Else

15 V\_bonus:=1500;

16 End if;

17 DBMS\_OUTPUT.PUT\_LINE('Employee Id:'||v\_emp\_id);

18 DBMS\_OUTPUT.PUT\_LINE('Employees bonus:'||v\_bonus);

19 END;

20 /

Enter value for employee\_id: 2

old 3: v\_emp\_id number(6):=&Employee\_ID;

new 3: v\_emp\_id number(6):=2;

Employee Id:2

Employees bonus:1500

PL/SQL procedure successfully completed.

1. Table Lecturer contains attributes as Lect\_Id, Major\_Sub, Course\_Name. Write a PLSQL block to fetch major subject of lecturer from

SQL> select \* from lecturer;

LECT\_ID MAJOR\_SUB COURSE\_NAME

---------- ---------- ----------

1 english en

2 economics eco

3 maths math

Declare

v\_major lecturer.major\_sub%type;

v\_course\_name varchar2(20);

begin

select major\_sub INTO v\_major

FROM lecturer

Where Lect\_Id=&Lecturer\_Id;

CASE v\_major

WHEN 'Computer' THEN

V\_course\_name:='CS101';

WHEN 'Economics' THEN

V\_course\_name:='EC203';

WHEN 'Physics' THEN

V\_course\_name:='PHY105';

ELSE

V\_course\_name:='UNKNOWN';

END CASE;

DBMS\_OUTPUT.PUT\_LINE('Major subject name is'||v\_course\_name);

End;

1. Write a PLSQL block which will take the marks of student from a user, calculate the percentage and decide the grade. If percentage is greater than 75 then its O grade if its between 75 and 60 then it’s A grade if its between 40 to 60 then its B grade else F.

Using case statement

Create table student(roll\_nonumber,sname varchar2(10),address varchar2(20),contact number(10),totalmarks number,percentage number, grade varchar2(10));

Declare

P1 number;

P2 number;

P3 number;

P4 number;

P5 number;

Total:=p1+p2+p3+p4+p5;

Per:=total/5;

Case

When per>75 then

g:=’O’;

When per>60 and per<75 then

g:=’A’;

When per>40 and per<60 then

g:=’B’;

else

g:=’F’;

end case;

insert into student values(701,’aaa’,’diva’,556,total,per,g);

end;

1. create a table attributes roomid, building, rm\_no, number\_seats, description. write a pl/sql block to comment on the type of place as fairly small,a little bigger,lots of room depending upon the value of number\_seats for a given room\_id.

DECLARE

v\_RoomId number:=&RoomId;

v\_No\_SeatsPlace.No\_Seats%TYPE;

BEGIN

Select no\_seats into v\_No\_Seats

From Place

Where Room\_Id=v\_RoomId;

If v\_no\_seats<3 Then

Dbms\_output.put\_line(‘fairly small room’);

elseifv\_no\_seats<=6 Then

Dbms\_output.put\_line(‘little bigger room’);

elseifv\_no\_seats<=10 Then

Dbms\_output.put\_line(‘lots of room’);

End if;

End;

1. **Writing PL/SQL Blocks with basic programming constructs for following Iterative Structure:**
2. **While-loop Statements**
3. **For-loop Statements.**
4. Create table circle

Write a PLSQL block to generate radius from 0 to all multiples of 5 till 50 and calculate area and perimeter to complete and save circle table**.**

SQL> Create table circle

2 (

3 Radius number(10,2),

4 Area number(10,2),

5 Perimeter number(10,2));

Table created.

SQL> declare

2 v\_radius circle.radius%type:=0;

3 v\_area circle.area%type:=0;

4 v\_perimeter circle.perimeter%type:=0;

5 begin

6 while v\_radius <= 50

7 loop

8 v\_area:=3.14\*v\_radius\*v\_radius;

9 v\_perimeter:=2\*3.14\*v\_radius;

10 insert into circle(radius,area,perimeter) values(v\_radius,v\_area,v\_perimete

r);

11 v\_radius:=v\_radius+5;

12 end loop;

13 end;

14 /

PL/SQL procedure successfully completed.

SQL> select \* from circle;

RADIUS AREA PERIMETER

---------- ---------- ----------

0 0 0

5 78.5 31.4

10 314 62.8

15 706.5 94.2

20 1256 125.6

25 1962.5 157

30 2826 188.4

35 3846.5 219.8

40 5024 251.2

45 6358.5 282.6

50 7850 314

11 rows selected.

Q. 2 Write PLSQL block to print series of odd numbers upto given numbers

DECLARE

v\_count number(2):=1;

v\_Num number(2):=&Number;

BEGIN

While v\_count<=v\_num

LOOP

Dbms\_output.put\_line(v\_count);

v\_count:=v\_count+2;

end loop;

END;

Q. 3 Pl/sql block to find factorial of number

Declare

n number(10):=&n;

f number(10,5);

begin

f:=1;

for i in 1..n

loop

f:=f\*i;

end loop;

dbms\_output.put\_line(‘factorial of nois’||f);

end;

1. **Writing PL/SQL Blocks with basic programming constructs by including a GoTO to jump out of a loop and NULL as a statement inside IF**

Q. Write a PLSQL program to check given number is prime number or not.

DECLARE

p number;

n number:=&number;

BEGIN

For i in 2..round(sqrt(n))

Loop

If n mod i = 0 then

P:=0;

GOTO print\_data;

END IF;

END LOOP;

P:=1;

<<print\_data>>

If p=1 then

DBMS\_OUTPUT.PUT\_LINE(TO\_CHAR(n)||’is prime number’);

Else

DBMS\_OUTPUT.PUT\_LINE(TO\_CHAR(n)||’is not prime number’);

End if;

END;

1. **Writing Procedures in PL/SQL Block**
2. **Create an empty procedure, replace a procedure and call procedure**
3. **Create a stored procedure and call it**
4. **Define procedure to insert data**
5. **A forward declaration of procedure**

A procedure is created with the **CREATE OR REPLACE PROCEDURE** statement. The simplified syntax for the CREATE OR REPLACE PROCEDURE statement is as follows −

CREATE [OR REPLACE] PROCEDURE procedure\_name

[(parameter\_name [IN | OUT | IN OUT] type [, ...])]

{IS | AS}

BEGIN

< procedure\_body >

END procedure\_name;

Where,

* *procedure-name* specifies the name of the procedure.
* [OR REPLACE] option allows the modification of an existing procedure.
* The optional parameter list contains name, mode and types of the parameters. **IN** represents the value that will be passed from outside and OUT represents the parameter that will be used to return a value outside of the procedure.
* *procedure-body* contains the executable part.
* The AS keyword is used instead of the IS keyword for creating a standalone procedure.

**1)**

CREATE OR REPLACE PROCEDURE greetings

AS

BEGIN

dbms\_output.put\_line('Hello World!');

END;

/

Procedure created.

EXECUTE greetings;

Hello World

PL/SQL procedure successfully completed.

The procedure can also be called from another PL/SQL block −

BEGIN

greetings;

END;

/

The above call will display −

Hello World

PL/SQL procedure successfully completed.

2) Program to find 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;

/

When the above code is executed at the SQL prompt, it produces the following result −

Minimum of (23, 45) : 23

PL/SQL procedure successfully completed.

**3) P**rocedure to 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;

/

When the above code is executed at the SQL prompt, it produces the following result −

Square of (23): 529

PL/SQL procedure successfully completed.

1. **Writing Functions in PL/SQL Block.**
2. **Define and call a function**
3. **Define and use function in select clause,**
4. **Call function in dbms\_output.put\_line**
5. **Recursive function**
6. **Count Employee from a function and return value back**
7. **Call function and store the return value to a variable**

A standalone function is created using the **CREATE FUNCTION** statement. The simplified syntax for the **CREATE OR REPLACE PROCEDURE** statement is as follows −

CREATE [OR REPLACE] FUNCTION function\_name

[(parameter\_name [IN | OUT | IN OUT] type [, ...])]

RETURN return\_datatype

{IS | AS}

BEGIN

< function\_body >

END [function\_name];

Where,

* *function-name* specifies the name of the function.
* [OR REPLACE] option allows the modification of an existing function.
* The optional parameter list contains name, mode and types of the parameters. IN represents the value that will be passed from outside and OUT represents the parameter that will be used to return a value outside of the procedure.
* The function must contain a **return** statement.
* The *RETURN* clause specifies the data type you are going to return from the function.
* *function-body* contains the executable part.
* The AS keyword is used instead of the IS keyword for creating a standalone function.

1. Example to illustrates how to create and call a standalone function. This function returns the total number of CUSTOMERS in the customers table.

Select \* from customers;

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

+----+----------+-----+-----------+----------+

CREATE OR REPLACE FUNCTION totalCustomers

RETURN number IS

total number(2) := 0;

BEGIN

SELECT count(\*) into total

FROM customers;

RETURN total;

END;

/

When the above code is executed using the SQL prompt, it will produce the following result −

Function created.

Following program calls the function **totalCustomers** from an anonymous block −

DECLARE

c number(2);

BEGIN

c := totalCustomers();

dbms\_output.put\_line('Total no. of Customers: ' || c);

END;

/

When the above code is executed at the SQL prompt, it produces the following result −

Total no. of Customers: 6

PL/SQL procedure successfully completed.

1. The following example demonstrates Declaring, Defining, and Invoking a Simple PL/SQL Function that computes and returns the maximum of two values.

DECLARE

a number;

b number;

c number;

FUNCTION findMax(x IN number, y IN number)

RETURN number

IS

z number;

BEGIN

IF x > y THEN

z:= x;

ELSE

Z:= y;

END IF;

RETURN z;

END;

BEGIN

a:= 23;

b:= 45;

c := findMax(a, b);

dbms\_output.put\_line(' Maximum of (23,45): ' || c);

END;

/

When the above code is executed at the SQL prompt, it produces the following result –

Maximum of (23,45): 45

PL/SQL procedure successfully completed.

1. **Writing a recursive Functions in PL/SQL Block**

A program or subprogram may call another subprogram. When a subprogram calls itself, it is referred to as a recursive call and the process is known as **recursion**.

To illustrate the concept, let us calculate the factorial of a number. Factorial of a number n is defined as −

n! = n\*(n-1)!

= n\*(n-1)\*(n-2)!

...

= n\*(n-1)\*(n-2)\*(n-3)... 1

The following program calculates the factorial of a given number by calling itself recursively −

DECLARE

num number;

factorial number;

FUNCTION fact(x number)

RETURN number

IS

f number;

BEGIN

IF x=0 THEN

f := 1;

ELSE

f := x \* fact(x-1);

END IF;

RETURN f;

END;

BEGIN

num:= 6;

factorial := fact(num);

dbms\_output.put\_line(' Factorial '|| num || ' is ' || factorial);

END;

/

When the above code is executed at the SQL prompt, it produces the following result −

Factorial 6 is 720

PL/SQL procedure successfully completed.

1. **Study of transactions and locks**

SQL> declare

var\_sal emp.sal%type;

begin

select sal into var\_sal

from emp

where empno=7844;

dbms\_output.put\_line('Salary:'||var\_sal||'(Original)');

update emp set sal=sal\*1.1;

--Increment salary by 10%

select sal into var\_sal

from emp

where empno=7844;

dbms\_output.put\_line('Salary'||var\_sal||'(Before Savepoint A)');

savepoint A;

update emp set sal=sal\*0.8;

select sal into var\_sal

from emp

where empno=7844;

dbms\_output.put\_line('salary:'||var\_sal||'(Savepoint A)');

savepoint B;

update emp set sal=sal\*1.3;

select sal into var\_sal

from emp

where empno=7844;

dbms\_output.put\_line('Salary:'||var\_sal||'(Savepoint B)');

rollback to savepoint B;

select sal into var\_sal

from emp

where empno=7844;

dbms\_output.put\_line('Salary:'||var\_sal||'(Rollback B)');

end;

/

**OUTPUT**

Salary:4646(Original)

Salary5111(Before Savepoint A)

salary:4089(Savepoint A)

Salary:5316(Savepoint B)

Salary:4089(Rollback B)

PL/SQL procedure successfully completed.

1. **Creating a Cursors in Pl/sql functions**

* Necessary table:

SQL> create table employee

2 (

3 emp\_id number(6),

4 lname varchar2(19

5 ),

6 fname varchar2(20),

7 birth\_date DATE

8 );

Table created.

SQL> desc employee;

Name Null? Type

----------------------------------------- -------- ----------------------------

EMP\_ID NUMBER(6)

LNAME VARCHAR2(19)

FNAME VARCHAR2(20)

BIRTH\_DATE DATE

SQL> set serveroutput on;

SQL> insert into employee values(1,'Cameron','John','03-Aug-1999');

1 row created.

SQL> insert into employee values(2,'Mark','Twain','19-Jan-1988');

1 row created.

SQL> insert into employee values(3,'Jhonson','John','29-May-2000');

1 row created.

SQL> insert into employee values(4,'Will','Smith','17-Oct-2001');

1 row created.

SQL> select \* from employee;

EMP\_ID LNAME FNAME BIRTH\_DAT

---------- ------------------- -------------------- ---------

1 Cameron John 03-AUG-99

2 Mark Twain 19-JAN-88

3 Jhonson John 29-MAY-00

4 Will Smith 17-OCT-01

* Cursor Implementation:

SQL> declare

2 old\_birth\_date DATE;

3 last\_name employee.lname%type;

4 first\_name employee.fname%type;

5 cursor find\_old\_bday is select min(birth\_date) from employee;

6 cursor emp\_id is select lname,fname from employee

7 where

8 birth\_date=old\_birth\_date;

9 begin

10 open find\_old\_bday;

11 fetch find\_old\_bday into old\_birth\_date;

12 close find\_old\_bday;

13 open emp\_id;

14 fetch emp\_id into last\_name,first\_name;

15 close emp\_id;

16 dbms\_output.put\_line('The oldest Employee of company is'||last\_name||first\_name);

17 end;

18 /

The oldest Employee of company isMarkTwain

PL/SQL procedure successfully completed.

1. **Creating packages in PL/SQL**

* **Required Table**

SQL> CREATE TABLE CUSTOMERS(

2 ID NUMBER(3) NOT NULL,

3 NAME VARCHAR (20) NOT NULL,

4 AGE NUMBER(3) NOT NULL,

5 ADDRESS VARCHAR2(25),

6 SALARY NUMBER(18, 2),

7 PRIMARY KEY (ID)

8 );

Table created.

SQL> desc customers;

Name Null? Type

----------------------------------------- -------- ----------------------------

ID NOT NULL NUMBER(3)

NAME NOT NULL VARCHAR2(20)

AGE NOT NULL NUMBER(3)

ADDRESS VARCHAR2(25)

SALARY NUMBER(18,2)

SQL> INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)

2 VALUES (1, 'Ramesh', 32, 'Ahmedabad', 2000.00 );

1 row created.

SQL>

SQL> INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)

2 VALUES (2, 'Khilan', 25, 'Delhi', 1500.00 );

1 row created.

SQL>

SQL> INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)

2 VALUES (3, 'kaushik', 23, 'Kota', 2000.00 );

1 row created.

SQL>

SQL> INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)

2 VALUES (4, 'Chaitali', 25, 'Mumbai', 6500.00 );

1 row created.

SQL>

SQL> INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)

2 VALUES (5, 'Hardik', 27, 'Bhopal', 8500.00 );

1 row created.

SQL>

SQL> INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)

2 VALUES (6, 'Komal', 22, 'MP', 4500.00 );

1 row created.

SQL> select \* from customers;

ID NAME AGE ADDRESS SALARY

---------- -------------------- ---------- ------------------------- ----------

1 Ramesh 32 Ahmedabad 2000

2 Khilan 25 Delhi 1500

3 kaushik 23 Kota 2000

4 Chaitali 25 Mumbai 6500

5 Hardik 27 Bhopal 8500

6 Komal 22 MP 4500

6 rows selected.

* **Package declaration**

SQL> CREATE PACKAGE cust\_sal AS

2 PROCEDURE find\_sal(c\_id customers.id%type);

3 END cust\_sal;

4 /

Package created.

* **Package body declaration:**

SQL> CREATE OR REPLACE PACKAGE BODY cust\_sal AS

2

3 PROCEDURE find\_sal(c\_id customers.id%TYPE) IS

4 c\_sal customers.salary%TYPE;

5 BEGIN

6 SELECT salary INTO c\_sal

7 FROM customers

8 WHERE id = c\_id;

9 dbms\_output.put\_line('Salary: '|| c\_sal);

10 END find\_sal;

11 END cust\_sal;

12 /

Package body created.

* **Implementing Package**

SQL> set serveroutput on;

SQL> DECLARE

2 code customers.id%type := &cc\_id;

3 BEGIN

4 cust\_sal.find\_sal(code);

5 END;

6 /

Enter value for cc\_id: 4

old 2: code customers.id%type := &cc\_id;

new 2: code customers.id%type := 4;

Salary: 6500

PL/SQL procedure successfully completed.

SQL>