Cycle-3

1. Write a PL/SQL block to do the following:
   * + 1. Read a number and print its multiplication table

**Program 1.1**

**declare**

**n number := &n;**

**i number;**

**begin**

**for i in 1..20 loop**

**dbms\_output.put\_line(n||' X '||i||' = '||n\*i);**

**end loop;**

**end;**

SQL> @ p31;

Enter value for n: 3

old 2: n number := &n;

new 2: n number := 3;

3 X 1 = 3

3 X 2 = 6

3 X 3 = 9

3 X 4 = 12

3 X 5 = 15

3 X 6 = 18

3 X 7 = 21

3 X 8 = 24

3 X 9 = 27

3 X 10 = 30

3 X 11 = 33

3 X 12 = 36

3 X 13 = 39

3 X 14 = 42

3 X 15 = 45

3 X 16 = 48

3 X 17 = 51

3 X 18 = 54

3 X 19 = 57

3 X 20 = 60

PL/SQL procedure successfully completed

2.Read a number and check whether it is a palindrome or not

**Program 1.2**

**declare**

**n number := &n;**

**rem number;**

**s number := 0;**

**temp number;**

**begin**

**temp := n;**

**while (n > 0) loop**

**rem := mod(n,10);**

**s:= s\*10+a;**

**n:= trunc(n/10);**

**end loop;**

**if temp = s then**

**dbms\_output.put\_line(temp||' is a palindrome');**

**else**

**dbms\_output.put\_line(temp||' is not a palindrome');**

**end if;**

**end;**

SQL> @p312;

19 /

Enter value for n: 1234

old 2: n number := &n;

new 2: n number := 1234;

1234 is not a palindrome

PL/SQL procedure successfully completed.

SQL> @p312;

19 /

Enter value for n: 9119

old 2: n number := &n;

new 2: n number := 9119;

9119 is a palindrome

PL/SQL procedure successfully completed.

3.Read a number n, and print the first n Fibonacci numbers

**Program 1.3**

**declare**

**a number;**

**b number;**

**c number;**

**n number;**

**i number;**

**begin**

**n:=&n;**

**a:=0;**

**b:=1;**

**dbms\_output.put\_line(a);**

**dbms\_output.put\_line(b);**

**for i in 1..n-2**

**loop**

**c:=a+b;**

**dbms\_output.put\_line(c);**

**a:=b;**

**b:=c;**

**end loop;**

**end;**

**OUTPUT:**

**SQL> @ FIBONACCI.sql**

**Enter value for n: 3**

**old   8: n:=&n;**

**new   8: n:=3;**

**0**

**1**

**1**

**PL/SQL procedure successfully completed.**

2.Write a PL/SQL block which will accept two numbers and find out their LCM and HCF. The output should be stored in a table called **DEMO\_TAB**.

**Program 2.**

**declare**

**a number := &a;**

**b number := &b;**

**g number;**

**l number;**

**a1 number;**

**b1 number;**

**begin**

**a1:=a;**

**b1:=b;**

**while (a <> b) loop**

**if (a < b) then**

**b := b - a;**

**else**

**a := a - b;**

**end if;**

**end loop;**

**g := a;**

**l:= (a1\*b1)/g;**

**insert into demo\_tab values(a1,b1,l,g);**

**end;**

SQL>); create table demo\_tab(n1 number,n2 number,lcm number,hcf number

Table created.

SQL> @p32;

22 /

Enter value for a: 4

old 2: a number := &a;

new 2: a number := 4;

Enter value for b: 8

old 3: b number := &b;

new 3: b number := 8;

PL/SQL procedure successfully completed.

SQL> select \* from demo\_tab;

N1 N2 LCM HCF

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

4 8 8 4

3.Consider the following relation schemas

Inventory

|  |  |  |
| --- | --- | --- |
| Product\_ID | Product\_name | Quantity |

Purchase\_Record

|  |  |  |
| --- | --- | --- |
| Product\_ID | Status | Date |

Write a PL/SQL block to read the quantity of a product from inventory and if it is > 0 reduce the quantity by 1 and record the status of purchase of that product as ‘PURCHASED’. Otherwise record the status of purchase of that product as ‘OUT OF STOCK’. While recording the status of a purchase, record the date of transaction.

**Program 3.**

**declare**

**id inven.pid%type := &id;**

**q inven.qty%type;**

**begin**

**select qty into q from inven where pid = id;**

**if q > 0 then**

**update inven set qty = qty - 1 where pid = id;**

**insert into pur\_rec values(id,'PURCHASED',sysdate);**

**else**

**insert into pur\_rec values(id,'OUTOFSTOCK',sysdate);**

**end if;**

**end;**

SQL> select \* from inven;

PID PNAME QTY

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

1 sorter 10

2 screw 0

3 bolt -10

SQL> select \* from pur\_rec;

no rows selected.

SQL> @p33;

13 /

Enter value for id: 1

old 2: id inven.pid%type := &id;

new 2: id inven.pid%type := 1;

PL/SQL procedure successfully completed.

SQL> select \* from inven;

PID PNAME QTY

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

1 sorter 9

2 screw 0

3 bolt -10

SQL> select \* from pur\_rec;

PID STATUS PDATE

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

1 PURCHASED 19-SEP-07

SQL> @p33;

13 /

Enter value for id: 2

old 2: id inven.pid%type := &id;

new 2: id inven.pid%type := 2;

PL/SQL procedure successfully completed.

SQL> select \* from inven;

PID PNAME QTY

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

1 sorter 9

2 screw 0

3 bolt -10

SQL> select \* from pur\_rec;

PID STATUS PDATE

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

1 PURCHASED 19-SEP-07

2 OUTOFSTOCK 19-SEP-07

4.Write a PL/SQL block to handle the following built-in exceptions

***no\_data\_found***

***too\_many\_rows***

***zero\_divide***

**Program 4.**

**declare**

**s employee.ssn%type ;**

**n number := 10;**

**m number;**

**begin**

**select ssn into s from employee;**

**m := n/0;**

**select ssn into s from employee where dno = 100;**

**exception**

**when too\_many\_rows then**

**dbms\_output.put\_line('too many rows returned where only 1 is expected');**

**when zero\_divide then**

**dbms\_output.put\_line('attempting to divide by zero');**

**when no\_data\_found then**

**dbms\_output.put\_line('no such department exists');**

**end;**

SQL> @p34;

14 /

too many rows returned where only 1 is expected

PL/SQL procedure successfully completed.

SQL> @p34;

14 /

attempting to divide by zero

PL/SQL procedure successfully completed.

SQL> @p34;

18 /

no such department exists

PL/SQL procedure successfully completed.

5 .Write a PL/SQL block to check whether the quantity of any product in Inventory table of problem 3 is < 0. If so, using an exception display relevant message and update quantity to 0.

**Program 5.**

**declare**

**id inven.pid%type ;**

**invalid\_qty exception;**

**begin**

**select pid into id from inven where qty < 0;**

**if sql%found then raise invalid\_qty;**

**end if;**

**exception**

**when no\_data\_found then**

**dbms\_output.put\_line('NO DATA FOUND');**

**when too\_many\_rows then**

**dbms\_output.put\_line('more than one row selected where only 1 is expected');**

**when invalid\_qty then**

**dbms\_output.put\_line('INVALID QUANTITY');**

**update inven set qty = 0 where pid = id;**

**end;**

SQL> select \* from inven;

PID PNAME QTY

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

1 sorter 10

2 screw -5

SQL> @p33;

15 /

INVALID QUANTITY

PL/SQL procedure successfully completed.

SQL> select \* from inven;

PID PNAME QTY

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

1 sorter 10

2 screw 0

1. Consider the following relation schemas

Emp

|  |  |  |  |
| --- | --- | --- | --- |
| Empid | name | salary | dno |

Del\_History

|  |  |  |
| --- | --- | --- |
| Dno | Rows\_deleted | date |

Write a PL/SQL block to delete records of all employees who belong to a particular department and then record the dno, no of rows deleted and date on which deletion occurred in the Del\_History table.

**Program 6.**

**declare**

**dn emp1.dno%type := &dn;**

**c number;**

**begin**

**delete from emp1 where dno = dn;**

**if sql%found then**

**c := sql%rowcount;**

**insert into del\_history values(dn,c,sysdate);**

**end if;**

**end;**

SQL> create table emp1 (empid number,name varchar2(12),salary number,dno number);

Table created.

SQL> select \* from emp1;

EMPID NAME SALARY DNO

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

1 aaaa 1111 10

2 bbbb 2222 10

3 cccc 2222 20

4 rrrr 3333 30

SQL> create table del\_history(dno number, rows\_deleted number, del\_date date);

Table created.

SQL> @p36;

11 /

Enter value for dn: 10

old 2: dn emp1.dno%type := &dn;

new 2: dn emp1.dno%type := 10;

PL/SQL procedure successfully completed.

SQL> select \* from emp1;

EMPID NAME SALARY DNO

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

3 cccc 2222 20

4 rrrr 3333 30

SQL> select \* from del\_history;

DNO ROWS\_DELETED DEL\_DATE

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

10 2 20-SEP-07

1. Consider the following relation schemas

Bank\_Main

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Acc\_no | Name | Address | Acc-type | Curr\_balance |

Bank\_Trans

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Acc\_no | Tr\_type | Tr\_date | Tr\_amt | Upd\_flag |

Where acc\_type is S – savings C- current

Tr\_type is D – deposit W – withdrawal

Write a PL/SQL block to update master table’s (i.e., Bank\_Main) curr\_balance field with transaction details from transaction file (i.e., Bank\_Trans) and also change the status of the Upd\_flag field of transaction table to ‘Y’ once the updation is complete.

**Program 7.**

**declare**

**a bank\_trans.acc\_no%type;**

**tt bank\_trans.trans\_type%type;**

**ta bank\_trans.tr\_amt%type;**

**cursor bank\_cursor is**

**select acc\_no,trans\_type,tr\_amt from bank\_trans where upd\_flag = 'n';**

**begin**

**open bank\_cursor;**

**loop**

**fetch bank\_cursor into a,tt,ta ;**

**exit when bank\_cursor%notfound ;**

**if tt = 'd' then**

**update bank\_main set open\_bal = open\_bal + ta where acc\_no = a ;**

**elsif tt = 'w' then**

**update bank\_main set open\_bal = open\_bal - ta where acc\_no = a;**

**end if;**

**update bank\_trans set upd\_flag = 'y' where acc\_no = a and trans\_type = tt;**

**end loop;**

**end;**

SQL> select \* from bank\_trans;

ACC\_NO TRANS TRANS\_DAT TR\_AMT U

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

1001 d 20-SEP-07 5000 n

1002 w 20-SEP-07 2000 y

1003 d 20-SEP-07 4000 y

1001 w 20-SEP-07 10000 n

SQL> select \* from bank\_main;

ACC\_NO NAME ADDRESS ACC\_T OPEN\_BAL

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

1001 sriya guntur s 15000

1002 bindu vizag s 10000

1003 abhilash hyderabad c 10000

1004 babu vijayawada c 9000

SQL> @p37;

22 /

PL/SQL procedure successfully completed.

SQL> select \* from bank\_main;

ACC\_NO NAME ADDRESS ACC\_T OPEN\_BAL

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

1001 sriya guntur s 10000

1002 bindu vizag s 10000

1003 abhilash hyderabad c 10000

1004 babu vijayawada c 9000

SQL> select \* from bank\_trans;

ACC\_NO TRANS TRANS\_DAT TR\_AMT U

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

1001 d 20-SEP-07 5000 y

1002 w 20-SEP-07 2000 y

1003 d 20-SEP-07 4000 y

1001 w 20-SEP-07 10000 y

Cycle-4

1. Write a PL/SQL stored function to calculate ncr using fact (m) function that returns factorial of m.

**PROGRAM 1:**

**FUNCTION FOR CALCULATING FACTORIAL**

CREATE or replace FUNCTION fact (n number) RETURN number AS

f number := 1;

a number := n;

BEGIN

WHILE (a > 1 ) LOOP

f := f \* a;

a := a - 1;

END LOOP;

RETURN f;

END;

**FUNCTION FOR CALCULATING NCR**

CREATE or replace FUNCTION ncr (n number, r number) RETURN number AS

ncr number ;

BEGIN

ncr:= trunc(fact(n)/(fact(r)\*fact(n-r)));

RETURN ncr;

END;

**MAIN PROGRAM**

declare

n number := &n;

r number := &r;

ncr1 number;

begin

ncr1 := ncr(n,r);

dbms\_output.put\_line(n||' c '||r||' = '||ncr1);

end;

1. Write a PL/SQL block that updates salary of an employee in employee table by using incr function which takes employee number as argument, calculates increment and returns increment based on the following criteria.

If salary <= 3000 – increment = 30% of salary

If salary > 3000 and <= 6000– increment = 20% of salary

Else increment = 10% of salary.

**PROGRAM 2:**

**FUNCTION FOR CALCULATING INCREMENT**

create or replace function incr(n number) return number as

s number;

begin

select sal into s from emp2 where empno=n;

if s <= 3000 then s:=0.3\*s;

elsif s>3000 and s<=6000 then

s:=0.2\*s;

else s:=0.1\*s;

end if;

return s;

end;

**MAIN PROGRAM**

declare

n number:=&no;

s number;

begin

s:=incr(n);

dbms\_output.put\_line('incr salary'||s);

update emp2 set sal=sal+s where empno=n;

end;

1. Write a stored procedure, raise\_salary which accepts an employee number. It uses incr function of previous program to get the salary increase amount and uses employee number to select the current salary from employee table. If employee number is not found or if the current salary is null, it should raise an exception. Otherwise, updates the salary.

**PROGRAM 3:**

**PROCEDURE FOR UPDATING SALARY**

create or replace procedure raisesal(n number) as

s number;

a number;

exc exception;

begin

s:=incr(n);

select sal into a from emp2 where empno=n;

if a is null then raise exc;

else update emp2 set sal=sal+s where empno=n;

end if;

exception

when no\_data\_found then

dbms\_output.put\_line('this employee is not exists');

when exc then

dbms\_output.put\_line('salary is null');

end;

**MAIN PROGRAM**

declare

n number:=&no;

r emp2%rowtype;

begin

raisesal(n);

select \* into r from emp2 where empno = n;

dbms\_output.put\_line(r.empno||’ ‘||r.sal);

end;

1. Consider the following table

ITEM

|  |  |  |  |
| --- | --- | --- | --- |
| prod\_ id | name | mat\_used | weight |

Write a procedure which will check for the existence of prod\_id from the table ITEM. This procedure must have two arguments, one of which will receive a value, which is a matching pattern for prod\_id of the ITEM table, and another which will return a value indicating whether a match has been found or not.qqqq

**PROGRAM 4:**

**PROCEDURE FOR CHECKING THE EXISTENCE OF A GIVEN PRODUCT ID**

create or replace procedure mat(n in number,s out char) as

a item.prod\_id%type;

begin

select prod\_id into a from item where prod\_id=n;

s:='match found';

exception

when no\_data\_found then

dbms\_output.put\_line('match not found');

end;

**MAIN PROGRAM**

declare

n number:=&id;

s char(12);

begin

mat(n,s);

dbms\_output.put\_line(s);

end;

1. Consider the following tables

PERSINFO

|  |  |  |
| --- | --- | --- |
| EMPNO | NAME | AGE |

AUDITPERSINFO

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| EMPNO | NAME | AGE | OPERATION | ODATE |

PERSINFO is the table for which the auditing must be performed and AUDITPERSINFO is the table which keeps track of the records deleted or modified. Create a database trigger audit\_trial. This trigger is forced when an UPDATE or a DELETE is performed on the table PERSINFO. It first checks for the operation being performed on the table. Then depending on the operation, a variable (that corresponds to operation) is assigned the value ‘UPDATE’ or ‘DELETE’ and then inserts the updated/deleted record into AUDITPERSINFO.

**PROGRAM 5:**

create or replace trigger audit\_trial

after update or delete on persinfo for each row

declare

eno number;

ename varchar2(12);

eage number;

op varchar2(12);

begin

if updating then op:='update';

elsif deleting then op:='delete';

end if;

eno := :old.empno;

ename := :old.name;

eage := :old.age;

insert into auditpersinfo

values(eno,ename,eage,op,sysdate);

end;