EX.NO:1

Creation of tables for Salesman and Customer Relation with following structure:

Salesman Relation:

salesman_id name City commission

Customer Relation:

customer_id cust_name City grade Saleman_id

Table creation:

SQL> create table salesman(salesman_id number(20),name varchar2(20),city varchar2(20), commission varchar2(20));

Table created.

SQL> create table customer(customer_id number(20),cust_name varchar2(20),city varchar2(20),grade varchar2(20),salesman_idva
rchar2(20));

Table created.

Alter command:

SQL> alter table salesman modify commission number(10);

Table altered.

SQL> insert into salesman values(5001,'james hoog','new york',0.15);

1 row created.

SQL> insert into salesman values(5002,'nail knite','paris',0.13);

1 row created.

SQL> insert into salesman values(5005,'pit alex','london',0.11);

1 row created.

SQL> insert into salesman values(5006,'mc lyon','paris',0.14);

1 row created.

SQL> insert into salesman values(5007,'paul adam','rome',0.13);

1 row created.

SQL> insert into salesman values(5003,'lauson hen','san jose',0.12);

1 row created.

SQL>desc salesman;

```
Name                                      Null?    Type
----------------------------------------- -------- ----------------------------
SALESMAN_ID                                        NUMBER(20)
NAME                                               VARCHAR2(20)
CITY                                               VARCHAR2(20)
COMMISSION                                         NUMBER(10)
```

Drop command:

SQL> alter table salesman drop column commission;

Table altered.

SQL>desc salesman;

```
Name                                      Null?    Type
----------------------------------------- -------- ----------------------------
SALESMAN_ID                                        NUMBER(20)
NAME                                               VARCHAR2(20)
CITY                                               VARCHAR2(20)
```

Truncate command:

SQL> **truncate table** salesman;

Table truncated.

SQL> **truncate table** customer;

Table truncated.

Rename command:

SQL>rename **table** salesman to salesmans;

Table renamed.

```
SQL>desc salesmans;

Name                                      Null?    Type
----------------------------------------- -------- ----------------------------
SALESMAN_ID                                        NUMBER(20)

NAME                                               VARCHAR2(20)

CITY                                               VARCHAR2(20)
```

.

EX.NO:2

Performing Insertion, Deletion, Modifying,Altering, Updating and Viewing records based on conditions.

SQL> create table salesman(salesman_id number(20),name varchar2(20),city varchar2(20), commission varchar2(20));

Table created.

SQL> create table customer(customer_id number(20),cust_name varchar2(20),city varchar2(20),grade varchar2(20),salesman_idva

rchar2(20));

Table created.

Insertion commands;

SQL> insert into salesman values(5001,'james hoog','new york',0.15);

1 row created.

SQL> insert into salesman values(5002,'nail knite','paris',0.13);

1 row created.

SQL> insert into salesman values(5005,'pit alex','london',0.11);

1 row created.

SQL> insert into salesman values(5006,'mc lyon','paris',0.14);

1 row created.

```
SQL> insert into salesman values(5007,'paul adam','rome',0.13);

1 row created.

SQL> insert into salesman values(5003,'lauson hen','san jose',0.12);

1 row created.

SQL>desc salesman;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 SALESMAN_ID                                        NUMBER(20)
 NAME                                               VARCHAR2(20)
 CITY                                               VARCHAR2(20)
 COMMISSION                                         NUMBER(10)
```

Update command:

```
SQL> update salesman set commission=0.13 where salesman_id=5002;

1 row updated.

SQL> update salesman set commission=0.11 where salesman_id=5005;

1 row updated.

SQL> update salesman set commission=0.14 where salesman_id=5006;
```

1 row updated.

SQL> update salesman set commission=0.13 where salesman_id=5007;

1 row updated.

SQL> update salesman set commission=0.12 where salesman_id=5003;

1 row updated.

Select commands:

SQL> select * from salesman;

```
SALESMAN_ID NAME                 CITY                 COMMISSION
----------- -------------------- -------------------- ----------
SALESMAN_ID NAME                 CITY                 COMMISSION
----------- -------------------- -------------------- --------------------
       5002 nail knite                paris                .13
       5005 pit alex                  london               .11
       5006 mc lyon                   paris                .14
       5007 pauladam                  rome                 .13
       5003 lauson hen                san jose             .12
       5001 jameshoog                 new york             .15
SQL> select name,city from salesman where commission > 0.13;
NAME                 CITY
-------------------- --------------------
mclyonparis
```

jameshoog new York

Delete command:

SQL> delete from salesman where commission=0.15 and name='';

0 rows deleted.

Alter and Modify Commands:

SQL> alter table salesman modify commission number(10);

Table altered.

SQL> alter table salesman drop column commission;

Table altered.

SQL> alter table salesman add commission varchar2(20);

Table altered.

SQL>desc salesman;

```
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 SALESMAN_ID                                        NUMBER(20)
 NAME                                               VARCHAR2(20)
 CITY                                               VARCHAR2(20)
 COMMISSION                                         VARCHAR2(20)
```

EX.NO:3

Creation of Views, Synonyms, Sequence, Save point and DCL

SQL> create table salesman(salesman_id number(20),name varchar2(20),city varchar2(20), commission varchar2(20));

Table created.

SQL> create table customer(customer_id number(20),cust_name varchar2(20),city varchar2(20),grade varchar2(20),salesman_idva

rchar2(20));

Table created.

View Creation using one table:

SQL> create or replace view fosi as select rollno,name from michael;

View created.

SQL>descfosi;

```
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ROLLNO                                              NUMBER(10)
 NAME                                                VARCHAR2(20)
```

SQL> select * from fosi;

```
   ROLLNO NAME
---------- --------------------
        20 josil
         1 fossi
```

View Creation using more than one table:

SQL>CREATE VIEW Salescust AS SELECT o.order_id, o.item, c.first_name, c.last_name FROM Customers c, Orders o WHERE o.Customer_id = c.Customer_id;

View created.

SQL> select * from salescust;

order_id	item	first_name	last_name
1	Keyboard	John	Reinhardt
2	Mouse	John	Reinhardt
3	Monitor	David	Robinson
4	Keyboard	John	Doe
5	Mousepad	Robert	Luna

Savepoint command:

SQL>savepoint salesman;

Savepoint created.

Sequence with no cycle:

SQL> create sequence sqrt start with 2 increment by 3 maxvalue 20 nocycle;

Sequence created.

SQL> insert into stud values(sqrt.nextval,'ddd','it',23,57);

1 row created.

SQL> select * from stud;

NO	NAME	NAMES	NUM	NUMB
2	ddd	it	23	57

SQL> insert into stud values(sqrt.nextval,'ddd','it',23,57);

1 row created.

SQL> select * from stud;

```
        NO NAME                NAMES                    NUM       NUMB
---------- -------------------- -------------------- ---------- ----------
         2 ddd                  it                       23         57
         5 ddd                  it                       23         57
```

Sequence with cycle:

SQL> create sequence sqrts start with 5 increment by 2 maxvalue 15 cycle cache 6;

Sequence created.

SQL> insert into stud values(sqrt.nextval,'ddd','it',sqrts.nextval,57);

1 row created.

SQL> insert into stud values(sqrt.nextval,'ddd','it',sqrts.nextval,57);

1 row created.

SQL> select * from stud;

```
        NO NAME                NAMES                    NUM       NUMB
---------- -------------------- -------------------- ---------- ----------
         2 ddd                  it                       23         57
         5 ddd                  it                       23         57
         8 ddd                  it                       23         57
        11 ddd                  it                        5         57
        14 ddd                  it                        7         57
```

Synonym command:

SQL> create synonym students for stud;

Synonym created.

SQL> select * from students;

NO	NAME	NAMES	NUM	NUMB
2	ddd	it	23	57
5	ddd	it	23	57
8	ddd	it	23	57
11	ddd	it	5	57
14	ddd	it	7	57
17	ddd	it	9	57
20	ddd	it	11	57
2	ddd	it	1	57
2	ddd	it	3	57
2	ddd	it	5	57
2	ddd	it	7	57

DATA CONTROL LANGUAGE (DCL)

CONNECTING TO DATABASE

SQL> connect system/manager@oracle11g;

Connected.

CREATING A NEW USER

SQL> create user sample identified by sample1;.

User created.

Enter user-name: sample

Enter password:

ERROR:

ORA-01045: user SAMPLENEW lacks CREATE SESSION privilege; logon denied

GRANT COMMAND

SQL> grant create session to sample;

Grant succeeded.

SQL> grant connect, resource to sample;

Grant succeeded.

LOGON TO NEW USER AFTER GIVING THE PERMISSION

Enter user-name: sample

Enter password:

Connected to:

Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - Production

With the Partitioning, OLAP, Data Mining and Real Application Testing options

ALL PRIVILEGES

SQL>grant all privileges on emp (table name) to sample (user name);Grant succeeded.

REVOKE ALL PRIVILEGES

SQL> revoke all privileges on emp (table name) from sample (user name);Revoke succeeded.

EX.NO:4

Set various constraints like Not Null,Primary Key, Foreign Key and Check constraints.

NOT NULL constraint:

SQL>CREATE TABLE Student(s_id int NOT NULL, name varchar(60), age int);

Table created.

SQL>insert into student value(200,’michael’,18);

1 row inserted

SQL>select * from student;

Student

s_id	name	age
200	michael	18

SQL>insert into student value(‘’,’michael’,18);

Error: NOT NULL constraint failed: Students.s-id

UNIQUE Constraint:

CREATE TABLE Student(s_id int NOT NULL, name varchar(60), age int NOT NULL UNIQUE);

SQL>insert into students values(200,'josil',18);

Students

s_id	name	age
200	josil	18

SQL>insert into students values(200,'josil',18);

Error: UNIQUE constraint failed: Students.age

Primary Key Constraint:

CREATE table Student (s_id int PRIMARY KEY, Name varchar(60) NOT NULL, Age int);

SQL>insert into studentss values(1001,'josil',18);

Studentss

s_id	Name	Age
1001	josil	18

SQL> insert into studentss values(1001,'josil',100);

Error: UNIQUE constraint failed: Studentss.s_id

Foreign Key Constraint:

CREATE table Order_Detail(order_id int PRIMARY KEY, order_name varchar(60) NOT NULL,c_id int FOREIGN KEY REFERENCES Customer_Detail(c_id));

Table created.

SQL>insert into order_details(1,’mich’,josil):

1 row inserted.

CHECK Constraint:

CREATE table Student(s_id int NOT NULL CHECK(s_id > 0),Name varchar(60) NOT NULL,Age int);

SQL> insert into studenty values(1001,'phenix',15);

Student

s_id	Name	Age
1001	phenix	15

SQL> insert into student values(0,'phenix',15);

Error: CHECK constraint failed: s_id > 0

EX.NO:5

Various join operations

1)SIMPLE JOIN

a) Equi-Join

SQL> select * from one,two where one.rollno=two.rollno;

```
   ROLLNO NAME                 ROLLNO     MARKS

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

         2 jeo                       2         89
```

b) Non Equi-Join

SQL> select * from one,two where one.rollno>=two.rollno;

```
   ROLLNO NAME                 ROLLNO     MARKS

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

         2 jeo                       2         89
```

2) SELF JOIN

SQL> select * from one o,two t where o.rollno=t.rollno;

```
   ROLLNO NAME                 ROLLNO     MARKS

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

         2 jeo                       2         89
```

3) INNER JOIN

SQL> select * from one inner join two using(rollno);

```
   ROLLNO NAME                 MARKS

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

         2 jeo                       89
```

4) NATURAL JOIN

SQL> select * from one natural join two;

```
   ROLLNO NAME                 MARKS
---------- -------------------- ----------
         2 jeo                  89
```

5) CROSS JOIN

SQL> select * from one cross join two;

```
   ROLLNO NAME                 ROLLNO     MARKS
---------- -------------------- ---------- ----------
         1 joo                  2         89
         1 joo                 34         99
         2 jeo                  2         89
         2 jeo                 34         99
```

6) OUTER JOIN

a) Left Outer Join

SQL> select * from one,two where one.rollno(+)=two.rollno;

```
   ROLLNO NAME                 ROLLNO     MARKS
---------- -------------------- ---------- ----------
         2 jeo                  2         89
                               34         99
```

b) Right Outer Join

SQL> select * from one,two where one.rollno=two.rollno(+);

```
   ROLLNO NAME                 ROLLNO     MARKS
---------- -------------------- ---------- ----------
         2 jeo                  2         89
         1 joo
```

c) Full Outer Join

SQL> select * from one full join two on one.rollno=two.rollno;

ROLLNO	NAME	ROLLNO	MARKS
2	jeo	2	89
1	joo	34	99

Different types of Joins:

1. SQL query to find the salesperson and customer whoreside in the same city. Return Salesman, cust_name and city

SQL> select cust_name, customer_id,city from customer where grade >=150 and grade <= 250;

CUST_NAME	CUSTOMER_ID	CITY
braddravis	3007	new york

2.SQL query to find those orders where the orderamount exists between 500 and 2000. Return ord_no, purch_amt, cust_name, city.

SQL> select cust_name,cudt_id,city from customer where grade>=150 and grade<=250;

CUST_NAME	CUSTOMER_ID	CITY
Brad dravis	3007	newyork

3.SQL query to find the salesperson(s) and thecustomer(s) he represents. Return Customer Name, city, Salesman, commission

SQL>select s.name,c.cust_name,s.commission from salesman s,custome c where s.salesman_id=c.sales_id;

NAME	CUST_NAME	COMMISSION
James hoog	nick rimando	.15
Nail knite	brad dravis	.15
Mc lyon	frabian	.14

4.SQL query to find salespeople who received commissions of more than 12 percent from the company. Return Customer Name, customer city, Salesman, commission.

SQL> select name,commission,city from salesman where commission>0.12;

NAME	COMMISSION	CITY
nailknite	.13	paris
mclyon	.14	paris
pauladam	.13	rome
jameshoog	.15	new York

5.SQL statement to join the tables salesman, customer and orders so that the samecolumn of each table appears once and only the relational rows are returned.

SQL>select s.salesman_id,s.name,c.cust_id,c.cust_name from salesman join customer c on s.salesman_id=c.salesman_id;

SALESMAN_ID	NAME	CUSTOMER_ID	CUSTOMER_NAME
5001	james hoog	3002	nick rimando
5001	james hoog	3007	brad dravis
5002	nail knite	3005	graham zusi
5006	mc lyon	3004	fabian

Ex.No.6

Simple PL/SQL Programs

(A) FACTORIAL OF A NUMBER

```
declare
n number(5);
i number(5);
fact number(20);
begin
n:=&number;
fact:=1;
for i in 1..n
loop
fact:=fact*i;
end loop;
dbms_output.put_line('Factorial of given number is'||fact);
end;
```

OUTPUT:

```
SQL> set serveroutput on
SQL> @fact.sql;
14 /
Enter value for number: 4
old   6:  n:=&number;
new   6:  n:=4;
Factorial of given number is24
PL/SQL procedure successfully completed.
```

(B) ODD OR EVEN

```
declare
n number(4);
begin
n:=&number;
if n mod 2=0 then
dbms_output.put_line('The number '||n||' is even');
else
dbms_output.put_line('The number '||n||' is odd');
end if;
end;
```

OUTPUT:

```
SQL> set serveroutput on
SQL> @odd.sql;
11 /
Enter value for number: 5
old   4:  n:=&number;
new   4:  n:=5;
The number 5 is odd
PL/SQL procedure successfully completed.
```

(C) FIBONACCI SERIES

```
declare
n number(3);
a number(3);
b number(3);
c number(3);
i number(3);
begin
n:=&number;
a:=-1;
b:=1;
for i in 1..n
loop
c:=a+b;
dbms_output.put_line(c);
a:=b;
b:=c;
end loop;
end;
```

OUTPUT:

```
SQL> set serveroutput on
SQL> @fib.sql;
 19  /
Enter value for number: 5
old   8:  n:=&number;
new   8:  n:=5;
0 1 1 2 3
PL/SQL procedure successfully completed.
```

(D) GREATEST OF THREE NUMBERS

```
declare
a number(5); b number(5); c number(5);
begin
a:=&number;
b:=&number;
c:=&number;
if a>b and a>c then
dbms_output.put_line('The bigger number is'||a);
else if b>c then
dbms_output.put_line('The bigger number is'||b);
else
dbms_output.put_line('The bigger number is'||c);
end if; end if; end;
```

OUTPUT:

```
SQL> set serveroutput on
SQL> @greatest.sql
 18  /
Enter value for number: 6
old   6: a:=&number;
new   6: a:=6;
Enter value for number: 4
old   7: b:=&number;
new   7: b:=4;
Enter value for number: 9
old   8: c:=&number;
new   8: c:=9;
The bigger number is9 PL/SQL procedure successfully completed.
```

(E) ARMSTRONG OR NOT

```
declare
n number(3):=&number;
digits number(5):=0;
arm number(5):=0;
n1 number(5);
begin
n1:=n;
while n>0
loop
digits:=mod(n,10);
arm:=arm+power(digits,3);
n:=trunc(n/10);
end loop;
if n1=arm then
dbms_output.put_line('The given number is a armstrong number');
else
dbms_output.put_line('The given number is not a armstrong number');
end if; end;
```

OUTPUT:

```
SQL> set serveroutput on
SQL> @armstrong.sql
 20  /
Enter value for number: 153
old   2: n number(3):=&number;
new   2: n number(3):=153;
The given number is a armstrong number
PL/SQL procedure successfully completed.
```

F) PALINDROME OR NOT

```
declare
n number(5);
num number(5);
r number(5);
rev number(5):=0;
begin
n:=&number;
num:=n;
while n>0 loop
r:=n mod 10;
rev:=(rev*10)+r;
n:=trunc(n/10);
end loop;
if rev=num then
dbms_output.put_line(' The given number is Palindrome');
else
dbms_output.put_line('The given number is Not Palindrome');
end if;
end;
```

OUTPUT:

```
SQL> /
Enter value for number: 252
old   7: n:=&number;
new   7: n:=252;
The given number is Palindrome
PL/SQL procedure successfully completed.
```

(G) PERFECT NUMBER

```
declare
n number(5);
i number(5);
f number(5):=0;
begin
n:=&number;
for i in 1..n-1
loop
if (n mod i=0)
then
f:=trunc(f+i);
end if;
end loop;
if (f=n) then
dbms_output.put_line('The given number is a perfect number ');
else
dbms_output.put_line('The given number is not a perfect number ');
end if;
end;
```

OUTPUT:

```
SQL> /
Enter value for number: 6
old   6: n:=&number;
new   6: n:=6;
The given number is a perfect number
PL/SQL procedure successfully completed.
```

Ex.no.7

Exception handling

Pre – defined Exception

Program -1:

```
declare
sno1  number;
mark1 number;
begin
select sno,mark into sno1,mark1 from t5 where sno=&sno1;
dbms_output.put_line('sno'|| sno1|| 'mark'||mark1);
exception
when no_data_found then
dbms_output.put_line('no data found');
when too_many_rows then
dbms_output.put_line('too many rows');
end;
```

OUTPUT

```
SQL> set serveroutput on

SQL> select * from t5;

      SNO       MARK
---------- ----------
       101        100
       103        100
       103        105
       105        105
       123         66

SQL> @ d:\excep.sql
Enter value for sno1: 103
old   5: select sno,mark into sno1,mark1 from t5 where sno=&sno1;
new   5: select sno,mark into sno1,mark1 from t5 where sno=103;
```

too many rows

```
PL/SQL procedure successfully completed.

SQL> /
```

```
SNO       MARK
---------- ----------
       101        100
       103        100
       103        105
       105        105
       123         66
SQL> @cc.sql
 13  /
Enter value for sno1: 102
old   5: select sno,mark into sno1,mark1 from t5 where sno=&sno1;
new   5: select sno,mark into sno1,mark1 from t5 where sno=102;
```

no data found

PL/SQL procedure successfully completed.

Program – 2:

```
declare
 num1 number;
 num2 number;
begin
 num1 := 10;
 num2 := 0;
 dbms_output.put_line('result:'||num1/num2);
exception
when zero_divide then
 dbms_output.put_line(sqlcode);
 dbms_output.put_line(sqlerrm);
end;
```

OUTPUT

```
SQL>  @ d:/zeroerr.sql
 15  /
-1476
ORA-01476: divisor is equal to zero
PL/SQL procedure successfully completed.
```

User –Defined Exception

Program -3:-

```
DECLARE
        myex EXCEPTION;
        i NUMBER;
BEGIN
        FOR i IN (SELECT * FROM stu) LOOP
                IF i.rollno = 103 THEN
                        RAISE myex;
                END IF;
        END LOOP;
EXCEPTION
        WHEN myex THEN
                dbms_output.put_line('Employee number already exist in student table.');
END;
```

OUTPUT

```
SQL> @ d:\useexp.sql
 14  /

Employee number already exist in student table.

PL/SQL procedure successfully completed.
```

Ex.no.8

PROCEDURE

Program 1:

WITHOUT PARAMETER

```
  create or replace procedure example is
     a number :=0;
    b number :=2;
      c number(2);
    begin
       c:=b/a;
   exception
    when zero_divide then
      dbms_output.put_line('divide by zero');
 * end example;
```

OUTPUT

```
SQL> exec example;
divide by zero
PL/SQL procedure successfully completed.
```

WITH IN AND OUT PARAMETER

Program 2:

```
//Factorial
 SQL> get z:\file\p1m.sql;
   create or replace procedure fact(n in number, f out number)
     is
       i number;
   begin
     f:=1;
```

```
    for i in 2..n loop
      f:=f*i;
     end loop;
  * end;
SQL> /
Procedure created.
//Program to call factorial procedure
SQL> get z:\file\p1s.sql;
    declare
     f number;
    n number:=&n;
    begin
     fact(n,f);
    dbms_output.put_line('Factorial of ' || n || 'is : ' || f);
  * end;
```

OUTPUT

```
SQL> /
Enter value for n: 5
old   3: n number:=&n;
new   3: n number:=5;
Factorial of 5is : 120
PL/SQL procedure successfully completed
```

Program -3

WITH IN OUT PARAMETER

```
//Procedure using inout parameter
SQL> get z:\p2m.sql;
  1  create or replace procedure doublen (n in out int) is
  2  begin
  3  n := n * 3;
  4* end;
SQL> /
Procedure created.
//Program to call procedure which use inout parameter
SQL> get z:\p2s.sql;
    declare
    r int;
    begin
    r := 7;
    dbms_output.put_line('before call r is: ' || r);
    doublen(r);
    dbms_output.put_line('after call r is: ' || r);
  * end;
```

OUTPUT

```
SQL> /
before call r is: 7
after call r is: 21
PL/SQL procedure successfully completed.
```

Ex.No.9a

FUNCTIONS

FACTORIAL OF A NUMBER

Program - 1

```
SQL> get z:\f2.sql;
    create or replace function factorial(a number) return number is
   f number:=1;
    i integer;
    begin
     for i in 1..a loop
        f:=i*f;
     end loop;
    return f;
  * end factorial;
SQL> /
Function created.
```

OUTPUT

```
SQL> select factorial(7) from dual;
FACTORIAL(7)
------------
        5040
```

Program – 2:

To display how many employees present in the employee database:-

```
SQL> select * from emp;

EMPID      ENAME                DEPT

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

101        arun                 cse

102        ravi                 cse

103        deva                 eee

103        varn                 it
```

Empf.sql

```
create or replace function empfn  return number is

total  number(2):=0;

begin

select count(*) into total from emp;

return total; end;

SQL> set serveroutput on;

SQL> @ d:/empf.sql

  8  /

Function created.
```

empmain.sql

```
declare

c number(2); begin

c:=empfn();

dbms_output.put_line('total employees:'||c); end;

SQL> @ d:/empmain.sql   7  /

total employees:4 PL/SQL procedure successfully completed.
```

Ex.No.9 b

TRIGGERS

1.PROGRAM

```
SQL> get z:\file\t.sql
  1  create or replace trigger dmlo
  2  after update or insert or delete on emp
  3  for each row
  4  begin
  5  if updating then
  6  dbms_output.put_line('table is updated');
  7  elsif inserting then
  8  dbms_output.put_line('table is inserted');
  9  elsif deleting then
 10  dbms_output.put_line('table is deleted');
 11  end if;
 12* end;
 13  /
Trigger created.
```

OUTPUT

```
SQL> set serveroutput on
SQL> select *from emp;

      ENO ENAME          BP        HRA        DA
---------- ------- ---------- ---------- ----------
       101 a          30000       100        50
       102 b          17000        70        30
       103 c          13000        50        25
SQL> insert into emp values(107,'e',13000,170,30);
```

table is inserted

1 row created.

SQL> select *from emp;

```
      ENO ENAME          BP        HRA        DA
---------- ------- ---------- ---------- ----------
      107 e          13000       170        30
      101 a          30000       100        50
      102 b          17000        70        30
      103 c          13000        50        25
```

SQL> delete from emp where eno='107';

table is deleted

1 row deleted.

SQL> select *from emp;

```
  ENO ENAME          BP        HRA        DA
---------- ------- ---------- ---------- ----------
      101 a          30000       100        50
      102 b          17000        70        30
      103 c          13000        50        25
```

SQL> update emp set bp=27000 where eno=101;

table is updated

1 row updated.

SQL> select *from emp;

```
      ENO ENAME          BP        HRA        DA
---------- ------- ---------- ---------- ----------
      101 a          27000       100        50
      102 b          17000        70        30
      103 c          13000        50        25
```

2.PROGRAM

SQL> get z:\t1.sql

```
  1  create trigger t1age
  2  before insert or update of age on trig
  3  for each row
  4  begin
  5  if(:new.age<0) then
  6  raise_application_error(-20000,'no negative age allowed');
  7  else
  8  dbms_output.put_line('valid age');
  9  end if;  10* end;
SQL> /
Trigger created.
```

OUTPUT

```
SQL> desc trig;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 NAME                                                VARCHAR2(7)
 AGE                                                 NUMBER(3)
SQL> insert into trig values('d',4);
```

valid age

```
1 row created.
SQL> insert into trig values('d',-4);
insert into trig values('d',-4)
            *
ERROR at line 1:
ORA-20000: no negative age allowed
ORA-06512: at "USER133.T1AGE", line 3 ORA-04088: error during execution of trigger
'USER133.T1AGE'
```

EX.NO.10

Database Connectivity with Front End Tools (Java/Python)

Java Database Connectivity

JDBC means access to the Java Database. It's a step forward for ODBC (Open Database Connectivity). JDBC is a standard API specification for moving data from the frontend to the backend. This API consists of classes and interfaces written in Java.

This simply serves as an interface between your Java system and databases (not the one we use in Java) or network, i.e. this provides a connection between the two so that a developer can send Java code information and store it in the database for future use.

The Java JDBC API allows Java applications to connect to relational databases such as MySQL, PostgreSQL, MS SQL Server, Oracle, H2 Database, etc. The JDBC API allows querying and updating relational databases, as well as calling stored procedures, and obtaining the database meta data.The Java JDBC API is part of the Java SE SDK core, making JDBC usable to all Java applications wishing to use it. Here is a diagram of a Java program connecting to a relational database using JDBC:

JDBC is independent from SQL

JDBC is not standardizing the SQL sent to the server. You, the JDBC API client, are writing the SQL. The SQL dialect used by the various databases varies slightly, so to be 100% independent of the database, you also need to be 100% independent of the database (i.e. use commands that are understood by all databases).

JDBC is not for databases that are not relational

The Java JDBC API is built to communicate with relational databases, meaning that you use standard SQL to connect with databases. The JDBC API is not intended for non-related servers like Mongo DB, Cassandra, Dynamo and so on. From a Java application you can use such databases, but you should see what drivers such databases provide for Java itself.

JDBC is independent of the type of database

The Java JDBC API standardizes how to connect to a database, how to execute queries against it, how to access a request output, how to execute database changes, how to call stored procedures,

and how to get meta data from the server. Through "standardizing," I mean the repository's software looks the same across various products.Therefore, if your project needs this in the future, it will be much easier to switch to another database.

Steps for Java program and database connectivity mentioned below:

- **Load JDBC driver**

First of all, you need to load or register the driver before you use it in the program. You must register once in your program. In one of the two ways listed below, you can register a driver:

- **Class.forName()**

Here at runtime we load the class file of the driver into memory. No need to use fresh or create object. The instance below uses Class.forName() to load the Oracle driver.–

Class.forName("org.h2.Driver");

- **DriverManager.registerDriver():**

DriverManager is an integrated Java class with a register of static members. Here we call the driver class constructor at the moment of compilation. DriverManager.registerDriver() is used in the following example/instance to register the Oracle driver -

DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver())

- **Connectionsestablisment**

We used code below after loading the driver to create connections:

Connection connection = DriverManager.getConnection(DBurl,username,password)

- **username :** it is a username from which your sql command prompt can be accessed.
- **password :** It is a password from which your sql command prompt can be accessed.
- **connection :** It is an element of connection, i.e. it is a communication interface reference.
- **DBurl :** It is a Uniform Resource Locator. It can be created as follows:

String url = “ jdbc:oracle:thin:@localhost:1521:xe”

Where Oracle is the database used, the driver used is tiny, where the database is located, @localhost is the IP address, 1521 is the port number, and xe is the service provider. The 3 parameters above are String sort, which the programmer must declare before calling the function. You may refer to the use of this from the final code.

- **Statement creation**

Whether the server needs to be modified or queried, you will need to build a JDBC Statement or JDBC PreparedStatement through which the update or request will occur.

Statement statement = connection.createStatement();

- **Query execution**

Here comes the most important part, i.e. the query execution. Here's a query from the SQL. We here understand that we can have various kinds of questions. Some of them are the following:

- **Request to delete / modify / insert a table in a database.**
- **Request to collect or retrieve information from the database.**

The Statement interface's executeQuery) (method is used to perform queries to extract server values. This method returns the ResultSet item which can be used to obtain from the table all data / records.

The Statement interface executeUpdate(sql query) method is used to perform update / insert.

Example:

```
int result = stmt.executeUpdate(sql);

if (result == 1)

System.out.println("inserted successfully : "+sql);

else

System.out.println("insertion failed");
```

Here sql is sql query of the type String

- **Close database connections**

You have to open the connection again when you're finished with the JDBC server connection. In the request, but also within the database server, a JDBC connection could take up a large amount of sources. Therefore, after use it is important to close the connection to the database again. You close a JDBC relation through the method of closing).

Here is an example of closing a JDBC connection:

```
connection.close();
```

Working example below :

```
importjava.sql.*;

importjava.util.*;
```

Java JDBC program below:

```
classMyFirstJDBCProgram
{
  public static void main(String a[])
  {
    String url = "jdbc:oracle:thin:@localhost:1521:xe";
    String username = "India";
    String password = "India";
    Scanner k = new Scanner(System.in);
System.out.println("enter class student name");
    String Studentname = k.next();
System.out.println("enter class student roll no");
    intstudentRollNumber = k.nextInt();
System.out.println("enter student current class");
    String Studentcls =  k.next();
    String sql = "insert into student1
values('"+Studentname+"','"+studentRollNumber+","'+Studentcls+"')";
    Connection conn=null;
    try
    {
DriverManager.registerDriver(new oracle.jdbc.OracleDriver());
      con = DriverManager.getConnection(url,user,pass);
      Statement stmt = conn.createStatement();
      int count = stmt.executeUpdate(sql);
      if (count == 1)
System.out.println("inserted student record successfully into database : "+sql);
      else
System.out.println("insertion failed.");
    }
```

```
    catch(Exception ex)
    {
System.err.println(ex);
    } finally {
conn.close();
    }
  }
}
```

How to connect Database in Python

The database is a well-organized collection of structured information or data stored in a computer system. In database, the data is arranged in the tabular form, and we can access that information or data by querying.

Python can be used to connect the Database. MySQL is one of the most popular Databases. In this tutorial, we will learn to establish a connection to MySQL via Python. Let's understand the following steps to work with the MySQL using Python.

1. Install MySQL Driver
2. Create a connection Object
3. Create a cursor Object
4. Execute the Query

Install MySQL Driver

First, we need a MySQL driver in our system. Install the MySQL software and configure the settings. We will use the MySQL connector driver, which is installed using the pip command. Open a command prompt and type the following command.

```
python -m pip install mysql-connector-python
```

Press the enter button. It will download the MySQL driver.

Verify the Driver

Let's check whether we have installed it properly or not. It can be done by importing the mysql.connector.

import mysql.connector

If this line is executed without error, it means MySQL connector has installed properly. We are ready to use it.

Create a Connection Object

The mysql.connector provides the **connect()** method used to create a connection between the MySQL database and the Python application. The syntax is given below.

Syntax:

Conn_obj= mysql.connector.connect(host = <hostname>, user = <username>, passwd = <passwor d>)

The connect() function accepts the following arguments.

- **Hostname -** It represents the server name or IP address on which MySQL is running.
- **Username -** It represents the name of the user that we use to work with the MySQL server. By default, the username for the MySQL database is root.
- **Password -** The password is provided at the time of installing the MySQL database. We don't need to pass a password if we are using the root.
- **Database -** It specifies the database name which we want to connect. This argument is used when we have multiple databases.

Consider the following example.

Example -

```
import mysql.connector
  # Creating a the connection object
conn_obj = mysql.connector.connect(host="localhost", user="root", passwd="admin123")
  # printing the connection object
print(conn_obj)
```

Output:

```
<mysql.connector.connection.MySQLConnection object at 0x7fb142edd780>
```

Create a Cursor Object

The connection object is necessary to create because it provides the multiple working environments the same connection to the database. The **cursor()** function is used to create the cursor object. It is import for executing the SQL queries. The syntax is given below.

Syntax:

Con_obj = conn.cursor()

Let's understand the following example.

Example -

```
import mysql.connector
  # Creating the connection object
conn_obj = mysql.connector.connect(host="localhost", user="root", passwd="admin123", databas
e="mydatabase")
  # printing the connection object
print(conn_obj)
  # creating the cursor object
cur_obj = conn_obj.cursor()
  print(cur_obj)
```

Output:

```
MySQLCursor: (Nothing executed yet)
```

Executes the Query

In the following example, we will create a database by executing the query. Let's understand the following example.

Example -

```
import mysql.connector
  # Creating the connection object
conn_obj = mysql.connector.connect(host="localhost", user="root", passwd="admin123")
  # creating the cursor object
cur_obj = conn_obj.cursor()
  try:
```

```
    # creating a new database using query
    cur_obj.execute("create database New_PythonDB")
    # getting the list of all the databases which will now include the new database New_PythonDB
      dbms = cur_obj.execute("show databases")
  except:
    conn_obj.rollback() # it is used if the operation is failed then it will not reflect in your database
  for x in cur_obj:
    print(x)
  conn_obj.close()
```

Output:

```
'information_schema',)
('javatpoint',)
('javatpoint1',)
(New_Pythondb)
('mydb',)
('mydb1',)
('mysql',)
('performance_schema',)
('testDB',)
```

EX.NO:11A

SIMPLE CALCULATOR

CODING

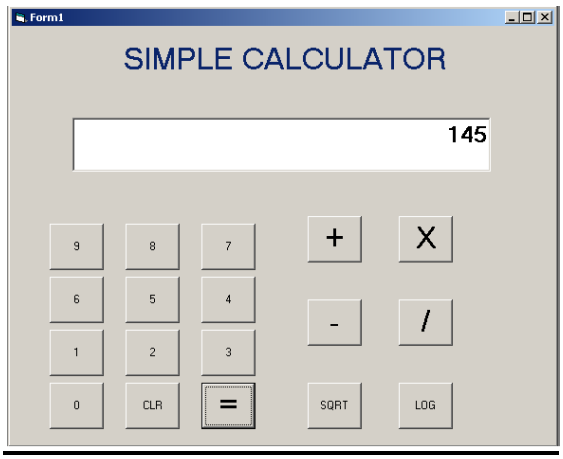
```
Dim Op1, Op2, Op3, Op As Variant
Private Sub Command1_Click()
R = Val(Text1.Text) * 10
R = R + 9
Text1.Text = R
End Sub
Private Sub Command10_Click()
R = Val(Text1.Text) * 10
R = R + 0
Text1.Text = R
End Sub
Private Sub Command11_Click()
Op1 = Val(Text1.Text)
Op2 = Math.Log(Op1)
Text1.Text = Op2
End Sub
Private Sub Command12_Click()
Text1.Text = " "
Op1 = Op2 = Op3 = Op = 0
End Sub
Private Sub Command13_Click()
Op1 = Val(Text1.Text)
Text1.Text = " "
Op = "*"
End Sub
```

```
Private Sub Command14_Click()
Op1 = Val(Text1.Text)
Text1.Text = " "
Op = "/"
End Sub
Private Sub Command15_Click()
Op1 = Val(Text1.Text)
Text1.Text = " "
Op = "-"
End Sub
Private Sub Command16_Click()
Op1 = Val(Text1.Text)
Text1.Text = " "
Op = "+"
End Sub
Private Sub Command17_Click()
Op2 = Val(Text1.Text)
If Op = "+" Then
Op3 = Op1 + Op2
ElseIf Op = "-" Then
Op3 = Op1 - Op2
ElseIf Op = "*" Then
Op3 = Op1 * Op2
ElseIf Op = "/" Then
Op3 = Op1 / Op2
Else
Text1.Text = "ERROR"
End If
Text1.Text = Op3
```

```
End Sub
Private Sub Command18_Click()
Op1 = Val(Text1.Text)
Op2 = Math.Sqr(Op1)
Text1.Text = Op2
End Sub
Private Sub Command2_Click()
R = Val(Text1.Text) * 10
R = R + 8
Text1.Text = R
End Sub
Private Sub Command3_Click()
R = Val(Text1.Text) * 10
R = R + 7
Text1.Text = R
End Sub
Private Sub Command4_Click()
R = Val(Text1.Text) * 10
R = R + 4
Text1.Text = R
End Sub
Private Sub Command5_Click()
R = Val(Text1.Text) * 10
R = R + 1
Text1.Text = R
End Sub
Private Sub Command6_Click()
R = Val(Text1.Text) * 10
R = R + 5
```

```
Text1.Text = R
End Sub
Private Sub Command7_Click()
R = Val(Text1.Text) * 10
R = R + 6
Text1.Text = R
End Sub
Private Sub Command8_Click()
R = Val(Text1.Text) * 10
R = R + 2
Text1.Text = R
End Sub
Private Sub Command9_Click()
R = Val(Text1.Text) * 10
R = R + 3
Text1.Text = R
End Sub
```

OUTPUT



EX.NO:11B

SIMPLE ARITHMETIC OPERATOR

Coding:

```
Private Sub Command1_Click(Index As Integer)
Select Case Index
        Case 0
        Text3.Text = Val(Text1.Text) + Val(Text2.Text)
        Case 1
        Text3.Text = Val(Text1.Text) - Val(Text2.Text)
        Case 2
        Text3.Text = Val(Text1.Text) * Val(Text2.Text)
        Case 3
        Text3.Text = Val(Text1.Text) / Val(Text2.Text)
        End Select
End Sub
```

Output