ExNo:1(A)	DDL Commands
Date:	

AIM:

To write and execute the Data Definition Language (DDL) Commands.

ALGORITHM:

Step1: Start executing the DDL Commands.

Step2: Using CREATE command create Employees table with EmployeeId as primary Key.

Step3: Using INSERT command insert the values into the Employees table.

Step4: Using ALTER command add a field in the table for Email.

Step5: Using TRUNCATE command delete the contents of the Employees table.

Step6: Using DROP command delete the structure of the table.

Step6: Stop the Process.

1. DDL Commands

PROGRAM:

--Create Table

```
SQL> CREATE TABLE Employees (EmployeeID Number(10)  PRIMARY KEY, FirstName
VARCHAR(50),  LastName VARCHAR(50),  Age Number(3), Department VARCHAR(50));

Table created.
```

--Insert Data into Table

```
SQL> INSERT INTO Employees (EmployeeID, FirstName, LastName, Age,
Department)VALUES          (1, 'John', 'Doe', 30, 'HR');

1 row created.

SQL> INSERT INTO Employees (EmployeeID, FirstName, LastName, Age,
Department)VALUES          (2, 'Jane', 'Smith', 28, 'IT');

1 row created.

SQL> INSERT INTO Employees (EmployeeID, FirstName, LastName, Age,
Department)VALUES          (3, 'Bob', 'Johnson', 35, 'Finance');

1 row created.

SQL> SELECT * FROM Employees;
```

OUTPUT:

EmployeeID	FirstName	LastName	Age	Department
1	John	Doe	30	HR
2	Jane	Smith	28	IT
3	Bob	Johnson	35	Finance

--Table Alter

```
SQL> ALTER TABLE Employees          ADD Email VARCHAR(100);

Table altered.
```

SQL> SELECT * FROM Employees;

EmployeeID	FirstName	LastName	Age	Department	Email
1	John	Doe	30	HR	
2	Jane	Smith	28	IT	
3	Bob	Johnson	35	Finance	

---TRUNCATE TABLE

SQL> TRUNCATE TABLE Employees;

Table truncated.

SQL> SELECT * FROM Employees;no

rows selected

--- DROP TABLE

SQL> DROP TABLE Employees;Table

dropped.

Result:

DDL Commands are executed successfully.

ExNo:1(B)	DML Commands
Date:	

AIM:

To write and execute Data Manipulation Language commands.

ALGORITHM:

Step1: Start Executing DML Commands.

Step2: Create a Students table with StudentId as primary key.

Step3: Insert the values into the student table.

Step4: Using UPDATE command modify an existing record.

Step5: Using DELETE command delete a particular record.

Step6: Stop the Process.

1. DML Commands

PROGRAM:

--Create Table

```
SQL> CREATE TABLE Students (StudentID Number(5)  PRIMARY KEY,FirstName VARCHAR(50),
                    LastName VARCHAR(50),          Age Number(3),Grade VARCHAR(2) );

Table created.
```

--Insert Data into Table

```
SQL> INSERT INTO Students (StudentID, FirstName, LastName, Age, Grade)VALUES(1,
   'Alice', 'Johnson', 20, 'A') ;

1 row created.

SQL> INSERT INTO Students (StudentID, FirstName, LastName, Age,
Grade)VALUES(2, 'Bob', 'Smith', 22, 'B');

1 row created.

SQL> INSERT INTO Students (StudentID, FirstName, LastName, Age, Grade) values(3, 'Charlie',
'Brown', 21, 'C');

1 row created.

SQL> SELECT * FROM Students;
```

STUDENTID	FIRSTNAME	LASTNAME	AGE	GR
1	Alice	Johnson	20	A
2	Bob	Smith	22	B
3	Charlie	Brown	21	C

--Update Table

```
SQL> UPDATE Students SET Grade = 'A' WHERE  StudentID = 2;

1 row updated.
```

SQL> SELECT * FROM Students;

STUDENTID	FIRSTNAME	LASTNAME	AGE	GR
1	Alice	Johnson	20	A
2	Bob	Smith	22	A
3	Charlie	Brown	21	C

-- INSERT INTO (additional record)

SQL> INSERT INTO Students (StudentID, FirstName, LastName, Age, Grade)VALUE(4, 'David', 'Williams', 23, 'B');

1 row created.

-- SELECT to view the new data

SQL> SELECT * FROM Students;

STUDENTID	FIRSTNAME	LASTNAME	AGE	GR
1	Alice	Johnson	20	A
2	Bob	Smith	22	A
3	Charlie	Brown	21	C
4	David	Williams	23	B

-- DELETE

SQL> DELETE FROM Students WHERE StudentID = 3;

1 row deleted.

SQL> SELECT * FROM Students;

STUDENTID	FIRSTNAME	LASTNAME	AGE	GR
1	Alice	Johnson	20	A
2	Bob	Smith	22	A
4	David	Williams	23	B

SQL> UPDATE StudentsSET AGE = 25 WHERE StudentID = 1;

1 row updated.

SQL> SELECT * FROM Students;

STUDENTID	FIRSTNAME	LASTNAME	AGE	GR
1	Alice	Johnson	25	A
2	Bob	Smith	22	A
4	David	Williams	23	B

Result:

DML commands are executed successfully.

ExNo:2	SQL SPECIAL OPERATORS
Date:	

AIM:

To write and execute Special Operators in SQL.

ALGORITHM:

Step1: Start the Process.

Step2: Create products table with productId as primary key.

Step3: Insert the values into the product table.

Step4: Execute like operator.

Step5: Execute IN operator.

Step6: Execute Between operator.

Step7: Execute Is null operator.

Step8: Execute Order by operator.

Step9: Stop the Process.

2.SQL SPECIAL OPERATORS

PROGRAM:

--Create Table

```
SQL> CREATE TABLE Products (ProductID Number(5)  PRIMARY KEY,ProductName
     VARCHAR(50),Price DECIMAL(8, 2),Category VARCHAR(50) );

     Table created.
```

--Insert Data into Table

```
SQL>  INSERT INTO Products (ProductID, ProductName, Price, Category)VALUES (1,
'Laptop', 1200.00, 'Electronics') ;

     1 row created.

SQL> INSERT INTO Products (ProductID, ProductName, Price, Category) V A L U E S  (2, 'Smartphone',
800.00, 'Electronics') ;

1 row created.

SQL> INSERT INTO Products (ProductID, ProductName, Price, Category) VALUES(3, 'Desk Chair',
150.00, 'Furniture');

    1 row created.

SQL> INSERT INTO Products (ProductID, ProductName, Price, Category) VALUES  (4, 'Coffee Table',
200.00, 'Furniture');

     1 row created.

SQL> INSERT INTO Products (ProductID, ProductName, Price, Category) VALUES  (5, 'Running Shoes',
80.00, 'Apparel');

1 row created.
```

-- SELECT using LIKE operator

```
SQL> SELECT * FROM Products  WHERE ProductName LIKE 'Desk%';
```

PRODUCTID	PRODUCTNAME	PRICE	CATEGORY
3	Desk Chair	150	Furniture

-

- SELECT using IN operator

SQL> SELECT * FROM Products WHERE Category IN ('Electronics', 'Furniture');

PRODUCTID	PRODUCTNAME	PRICE	CATEGORY
1	Laptop	1200	Electronics
2	Smartphone	800	Electronics
3	Desk Chair	150	Furniture
4	Coffee Table	200	Furniture

-- SELECT using BETWEEN operator

SQL> SELECT * FROM Products WHERE Price BETWEEN 100.00 AND 500.00;

PRODUCTID	PRODUCTNAME	PRICE	CATEGORY
3	Desk Chair	150	Furniture
4	Coffee Table	200	Furniture

-- SELECT using IS NULL operator

SQL> SELECT * FROM Products WHERE Category IS NULL;

no rows selected

-- SELECT with ORDER BY

SQL> SELECT * FROM Products ORDER BY Price DESC;

PRODUCTID	PRODUCTNAME	PRICE	CATEGORY
1	Laptop	1200	Electronics
2	Smartphone	800	Electronics
3	Desk Chair	150	Furniture
4	Coffee Table	200	Furniture
5	Running Shoes	80	Apparel

Result:

SQL Special operators were executed successfully.

ExNo:3	AGGREGATE FUNCTIONS
Date:	

AIM:

To write and execute Aggregate function using SQL.

ALGORITHM:

Step1: Start the Process.

Step2: Create sales Table with SalesId as a primary key.

Step3: Insert values into sales table.

Step4: Execute COUNT aggregate function.

Step5: Execute SUM aggregate function.

Step6: Execute AVG aggregate function.

Step7: Execute MIN aggregate function.

Step8: Execute MAX aggregate function.

Step9: Stop the Process.

3.AGGREGATE FUNCTIONS

PROGRAM:

-- CREATE TABLE

```
SQL> CREATE TABLE Sales (SaleID Number (15) PRIMARY KEY,ProductName
VARCHAR(50),Quantity INT,Price DECIMAL(8, 2));

Table created.
```

-- INSERT INTO

```
SQL> INSERT INTO Sales (SaleID, ProductName, Quantity, Price) VALUES(1, 'Laptop', 2, 1200.00);

1 row created.

SQL> INSERT INTO Sales (SaleID, ProductName, Quantity, Price)  VALUES(2, 'Smartphone', 3,
  800.00);

1 row created.

SQL> INSERT INTO Sales (SaleID, ProductName, Quantity, Price)  VALUES (3, 'Desk Chair', 1,
  150.00);

1 row created.

SQL> INSERT INTO Sales (SaleID, ProductName, Quantity, Price) VALUES(4, 'Coffee Table', 2,
  200.00);

1 row created.

SQL> INSERT INTO Sales (SaleID, ProductName, Quantity, Price)  VALUES(5, 'Running Shoes', 5,
  80.00);

1 row created.

SQL> Select * from Sales;
```

SALEID	PRODUCTNAME	QUANTITY	PRICE
1	Laptop	2	1200
2	Smartphone	3	800
3	Desk Chair	1	150
4	Coffee Table	2	200
5	Running Shoes	5	80

-- SELECT with COUNT

```
SQL> SELECT COUNT(*) AS TotalSales FROM Sales;

TOTALSALES
----------

         5
```

-- SELECT with SUM

```
SQL> SELECT SUM(Quantity) AS TotalQuantity, SUM(Price) AS TotalRevenueFROM Sales;

TOTALQUANTITY              TOTALREVENUE
-------------              ------------

           13                      2430
```

-- SELECT with AVG

```
SQL> SELECT AVG(Price) AS AveragePrice FROM Sales;

AVERAGEPRICE
------------

         486
```

-- SELECT with MIN

```
SQL> SELECT MIN(Price) AS MinPrice FROM Sales;

MINPRICE
----------

        80
```

-- SELECT with MAX

```
SQL> SELECT MAX(Price) AS MaxPrice FROM Sales;

MA XPRICE
----------

      1200
```

Result:

Aggregate functions are executed successfully.

ExNo:4	FUNCTIONS
Date:	

AIM:

To write and execute functions using SQL.

ALGORITHM:

Step1: Start the process.

Step2: Create Employees table with EmployeeId as primary key.

Step3: Insert values into Employee table.

Step4: Execcute built-in function ROUND().

Step5: Execute built-in function Case.

Step6: Execute user defined function to find minimum of two numbers.

Step7: Define find min function.

Step8: Begin the function declare two input variable and one output variable.

Step 9: Compare the input values and return the outputvalue and end function.

Step10: Begin the calling function.

Step 11: Give the values for input variables and print the output value.

Step12: End the calling function.

Step13: Stop the Process.

4.FUNCTIONS

PROGRAM:

-- CREATE TABLE

```
SQL>     CREATE TABLE Employees ( EmployeeID Number(15)  PRIMARY KEY, FirstName
VARCHAR(50), LastName VARCHAR(50), Salary DECIMAL(10, 2));

Table created.
```

-- INSERT INTO

```
SQL> INSERT INTO Employees (EmployeeID, FirstName, LastName, Salary)
VALUES(1, 'John', 'Doe', 50000.50);

1 row created.

SQL> INSERT INTO Employees (EmployeeID, FirstName, LastName, Salary)VALUES(2, 'Jane', 'Smith',
   60000.75);

1 row created.

SQL> INSERT INTO Employees (EmployeeID, FirstName, LastName, Salary)VALUES(3, 'Bob',
  'Johnson', 75000.25);

1 row created.

SQL> INSERT INTO Employees (EmployeeID, FirstName, LastName, Salary)VALUES(4, 'Sam', 'John',
  90000.25);

1 row created.
```

-- SELECT with mathematical functions

```
SQL> SELECT       ROUND v(Salary, 1) AS RoundedSalary,  ABS(Salary) AS
AbsoluteSalary FROM Employees;

ROUNDEDSALARY          ABSOLUTESALARY
-----------------------------------
    50000.5                 50000.5
    60000.7                 60000.75
    75000.2                 75000.25
    90000.2                 90000.25
```

-- SELECT with custom function using CASE

SQL> SELECT FirstName, LastName, Salary, CASE WHEN Salary < 60000 THEN 'Low' WHEN Salary >= 60000 AND Salary < 80000 THEN 'Medium' ELSE 'High' END AS SalaryCategory FROM Employees;

FIRSTNAME	LASTNAME	SALARY	SALARY
John	Doe	50000.5	Low
Jane	Smith	60000.75	Medium
Bob	Johnson	75000.25	Medium
Sam	John	90000.25	High

User Defined functions:

```
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;
/
```

Output:

Maximum of (23,45): 45

PL/SQL procedure successfully completed.

Result:

The function was executed successfully in SQL.

ExNo:5	SQL JOINS
Date:	

AIM:

To create and execute SQL joins using SQL.

ALGORITHM:

Step1: Start the Process.

Step2: Create two tables states and cities.

Step3: Insert values into the tables.

Step4: Execute INNER JOIN.

Step5: Execute LEFT JOIN.

Step6: Execute RIGHT JOIN.

Step7: Execute FULL OUTER JOIN.

Step8: Stop the Process.

5.SQL JOINS

PROGRAM:

--Table Create

SQL> CREATE TABLE States (StateID Number(15) PRIMARY KEY, StateName VARCHAR(50));

Table created.

SQL> CREATE TABLE Cities (CityID Number (15) PRIMARY KEY, CityName VARCHAR(50), StateID Number(15), FOREIGN KEY (StateID) REFERENCES States (StateID));

Table created.

-- INSERT DATA into States table

SQL> INSERT INTO States (StateID, StateName) VALUES (1, 'Maharashtra');1

row created.

SQL> INSERT INTO States (StateID, StateName) VALUES (2, 'Gujarat');1

row created.

SQL> INSERT INTO States (StateID, StateName) VALUES (3, 'Tamil Nadu');1

row created.

-- INSERT DATA into Cities table

SQL> INSERT INTO Cities (CityID, CityName, StateID) VALUES (101, 'Mumbai', 1);1

row created.

SQL> INSERT INTO Cities (CityID, CityName, StateID) VALUES (102, 'Pune', 1);1

row created.

SQL> INSERT INTO Cities (CityID, CityName, StateID) VALUES (103, 'Ahmedabad', 2);1 row created.

SQL> INSERT INTO Cities (CityID, CityName, StateID) VALUES (104, 'Surat', 2);1

row created.

SQL> INSERT INTO Cities (CityID, CityName, StateID) VALUES (105, 'Chennai', 3);

1 row created.
SQL> INSERT INTO Cities (CityID, CityName, StateID) VALUES (106, 'Coimbatore', 3);1 row created.

-- INNER JOIN

SQL> SELECT Cities.CityName, States.StateName FROM Cities INNER JOIN StatesON Cities.StateID = States.StateID;

CITYNAME	STATENAME
Mumbai	Maharashtra
Pune	Maharashtra
Ahmedabad	Gujarat
Surat	Gujarat
Chennai	Tamil Nadu
Coimbatore	Tamil Nadu6

rows selected.

-- LEFT JOIN

SQL> SELECT Cities.CityName, States.StateName FROM Cities LEFT JOIN States ONCities.StateID = States.StateID;

CITY NAME	STATE NAME
Mumbai	Maharashtra
Pune	Maharashtra
Ahmedabad	Gujarat
Surat	Gujarat
Chennai	Tamil Nadu
Coimbatore	Tamil Nadu6

rows selected.

-- RIGHT JOIN

SQL> SELECT Cities.CityName, States.StateName FROM Cities RIGHT JOIN StatesON Cities.StateID = States.StateID;

CITYNAME	STATENAME
Mumbai	Maharashtra
Pune	Maharashtra
Ahmedabad	Gujarat
Surat	Gujarat
Chennai	Tamil Nadu
Coimbatore	Tamil Nadu6

rows selected.

-- FULL OUTER JOIN (Not supported in all databases)

SQL> SELECT Cities.CityName, States.StateName 2 FROM Cities3 FULL OUTER JOIN States ON Cities.StateID = States.StateID;

CITYNAME	STATENAME
Mumbai	Maharashtra
Pune	Maharashtra
Ahmedabad	Gujarat
Surat	Gujarat
Chennai	Tamil Nadu
Coimbatore	Tamil Nadu

rows selected.

Result:

SQL Joins were executed successfully.

ExpNo:6	SUB QUERIES
Date:	

AIM:

To write and execute sub queries using SQL.

ALGORITHM:

Step1: Start the Process.

Step2: Create Countries and cities table.

Step2: Insert values into the tables.

Step3: Execute the subquery.

Step4: Stop the Process.

6.SUB QUERIES

PROGRAM

-- CREATE TABLES

```
SQL> CREATE TABLE Countries (CountryID Number(15)  PRIMARY KEY,
CountryNameVARCHAR(50) );

SQL> CREATE TABLE CITIES (CityID Number(15) PRIMARY KEY, CityName
VARCHAR(50),CountryID Number(15), Population Number(15), FOREIGN KEY (CountryID)
REFERENCES Countries(CountryID) );
```

-- INSERT DATA into Countries table

```
SQL> INSERT INTO Countries (CountryID, CountryName) VALUES (1, 'UnitedStates');

1 row created.

SQL> INSERT INTO Countries (CountryID, CountryName) VALUES (2, 'United Kingdom');

1 row created.

SQL> INSERT INTO Countries (CountryID, CountryName) VALUES  (3, 'India');1 row created.
```

-- INSERT DATA into Cities table

```
SQL> INSERT INTO Cities (CityID, CityName, CountryID, Population) VALUES(101, 'New York',
    1, 8500000);

1 row created.

SQL> INSERT INTO Cities (CityID, CityName, CountryID, Population) VALUES(102, 'London', 2,
    8200000);

1 row created.

SQL> INSERT INTO Cities (CityID, CityName, CountryID, Population) VALUES(103, 'Mumbai', 3,
 12400000);

1 row created.
```

```
SQL> INSERT INTO Cities (CityID, CityName, CountryID, Population) VALUES(104, 'Los
 Angeles', 1, 3980000);
1 row created.

SQL> INSERT INTO Cities (CityID, CityName, CountryID, Population) VALUES(105, 'Delhi', 3,
 28700000);

1 row created.

SQL> INSERT INTO Cities (CityID, CityName, CountryID, Population) VALUES(106,
 'Manchester', 2, 550000);

1 row created.
SQL> select * from Countries;

 COUNTRYID        COUNTRYNAME
---------------------  ---------------------------
         1              United States
         2              United Kingdom
         3               India

SQL> select * from Cities;

    CITYID    CITYNAME            COUNTRYID              POPULATION
 ----------------------------------------------------------------

       101        New York              1                    8500000

       102        London                2                     8200000

       103        Mumbai                3                    12400000

       104        Los Angeles           1                    3980000

       105        Delhi                 3                    28700000

       106        Manchester            2                     550000
```

-- SUBQUERY TO GET COUNTRIES WITH POPULATION GREATER THAN

```
SQL> SELECT CountryName 2  FROM Countries  3  WHERE CountryID IN (SELECT
    CountryID                    FROM Cities          WHERE Population > (SELECT
AVG(Population) FROM Cities) );

COUNTRYNAME
--------------------
India
```

Result:

Sub queries are executed successfully.

ExpNo:7	SEQUENCES AND VIEWS
Date:	

AIM:

To write and execute Sequence and Views using SQL.

ALGORITHM:

Step1: Start the process.

Step2: Create Employee table and insert values into the table.

Step3: Create SEQUENCE as emp_seq and insert values.

Step4: Create VIEW as Employee summary and display the view using select command.

Step5: Stop the process.

7. SEQUENCES AND VIEWS

PROGRAM:

--Create Table

```
SQL> CREATE TABLE EMPLOYEE ( EmployeeID Number(15) PRIMARY KEY,
FirstName VARCHAR(50), LastName VARCHAR(50), DepartmentVARCHAR(50),
Salary DECIMAL(10, 2) );

Table created.
```

--Insert Data into Table

```
SQL> INSERT INTO EMPLOYEE (EmployeeID, FirstName, LastName, Department,Salary) VALUES
(1, 'John', 'Doe', 'HR', 50000.00);

1 row created.

SQL> INSERT INTO EMPLOYEE (EmployeeID, FirstName, LastName, Department,
Salary)VALUES        (2, 'Jane', 'Smith', 'IT', 60000.00);

1 row created.

SQL> INSERT INTO EMPLOYEE (EmployeeID, FirstName, LastName, Department,
Salary)VALUES           (3, 'Bob', 'Johnson', 'Finance', 75000.00);

1 row created.
```

--Sequence command

```
SQL> CREATE SEQUENCE emp_sequence
   START WITH 1001          INCREMENT BY 1  NOCACHE            NOCYCLE;

Sequence created.

SQL> INSERT INTO EMPLOYEE (EmployeeID, FirstName, LastName, Department,Salary)
VALUES                  (emp_sequence.NEXTVAL, 'Alice', 'Williams', 'Marketing', 55000.00);

1 row created.

SQL> CREATE VIEW EmployeeSummary AS  SELECT EmployeeID, FirstName,LastName, Department,
Salary  FROM EMPLOYEE  WHERE Salary > 60000.00;

View created.
```

SQL> SELECT * FROM EmployeeSummary;

EMPLOYEEID	FIRSTNAME	LASTNAME	DEPARTMENT	SALARY
3	Bob	Johnson	Finance	75000

Result:

Sequence and views were created successfully in SQL.

ExNo:8	EXCEPTION HANDLING
Date:	

AIM:

To write and execute Exception Handling using SQL.

ALGORITHM:

Step1: Start the Process.

Step2: Declare the variables a,b ,answer and assign the values for a and b.

Step3: Begin the process, Divide a by b.

Step 4: If b is equal to zero exception will be called and the error message is displayed as output.

Step 5: End the process.

Step6: Stop the process.

8.EXCEPTION HANDLING

PROGRAM:

```
DECLARE
a int:=10;
b int:=0;
answer int;

BEGIN
answer:=a/b;
dbms_output.put_line('the result after division is'||answer);

Exception
WHEN zero_divide THEN
dbms_output.put_line('dividing by zero please check the values again');
dbms_output.put_line('the value of a is '||a);
dbms_output.put_line('the value of b is '||b);
END;
/
```

OUTPUT:

```
dividing by zero please check the values again
the value of a is 10
the value of b is 0
```

Result:

Exception handling was executed successfully using SQL.

ExNo:9	TRIGGERS
Date:	

AIM:

To write and execute Triggers using SQL.

ALGORITHM:

Step1: Start the process.

Step2: Create a table customer and insert the values.

Step3: Create triggers to insert and modify the customer table.

Step 4: Insert and update the customer table, trigger is triggered and result will be displayed.

Step 5: Stop the process.

9.TRIGGERS

Create table customers(ID Number(5) ,NAME varchar(15),AGE Number(5),ADDRESS varchar(25),SALARY Number(5)

PROGRAM:

Creating a Trigger:

```
Set serveroutput on;
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;
/
```

Output:
Trigger created.

Triggering a Trigger

```
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (7, 'Kriti', 22, 'HP', 7500.00 );
```

Output:
Old salary:
New salary: 7500
Salary difference:

```
UPDATE customers
SET salary = salary + 500
WHERE id = 2;
```

Output:
Old salary: 1500
New salary: 2000
Salary difference: 50

Result:

Triggers were successfully executed.

ExNo:10	PROCEDURES
Date:	

Aim:

To execute procedures using SQL.

Algorithm:

Step1: Start the process

Step2: Declare three variables a, b,c.

Step3: Define the procedure findmin and pass x and y variable as input variable and z as output variable.

Step4: Begin the procedure.

Step5: If x is less than y then z=x else z=y.

Step6: End the procedure.

Step7: Call the procedure findmin.

Step8: Begin the calling procedure.

Step9: Initialize the values for input variables.

Step10: Call the function findmin.

Step11: The minimum number was found .

Step12: Stop the process.

10. Procedures

Program:

```
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;
/
```

Output:

```
Minimum of (23, 45) : 23


PL/SQL procedure successfully completed.
```

Result:

Procedure was executed Successfully.