MAHA BARATHI ENGINEERING COLLEGE

NH-79, SALEM-CHENNAI HIGHWAY, A.VASUDEVANUR, CHINNASALEM TK, KALLAKURICHI DT – 606 201.

Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

2(f) & 12(B) status of UGC, New Delhi,

www.mbec.ac.in | 04151-256333, 257333 | mbec123@gmail.com

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CS3481 – DATABASE MANAGENT SYSTEMS LAB MANUAL

II Year/IV Semester B.E CSE

Regulation 2021
(As Per Anna University, Chennai syllabus)

PREPARED BY,
A.ANISHA DARATHY(AP/CSE)

VERIFIED BY,
Mr. N. KATHIRKUMAR (HOD / CSE)

CONTENTS

Subject Code : CS3481
Subject Name : DATABASE MANAGEMENT SYSTEMS LABORATORY
Year & Semester : II / IV

CS3481 / DATABASE MANAGEMENT SYSTEMS LABORATORY

LIST OF EXPERIMENTS

COURSE OBJECTIVES:

- To learn and implement important commands in SQL.
- To learn the usage of nested and joint queries.
- To understand functions, procedures and procedural extensions of databases.
- To understand design and implementation of typical database applications.
- To be familiar with the use of a front end tool for GUI based application development.

LIST OF EXPERIMENTS:

1. Create a database table, add constraints (primary key, unique, check, Not null), insert rows, update and delete rowsusing SQL DDL and DML commands.
2. Create a set of tables, add foreign key constraints and incorporate referential integrity.
3. Query the database tables using different 'where' clause conditions and also implement aggregate functions.
4. Query the database tables and explore sub queries and simple join operations.
5. Query the database tables and explore natural, equi and outer joins.
6. Write user defined functions and stored procedures in SQL.
7. Execute complex transactions and realize DCL and TCL commands.
8. Write SQL Triggers for insert, delete, and update operations in a database table.
9. Create View and index for database tables with a large number of records.
10. Create an XML database and validate it using XML schema.
11. Create Document, column and graph based data using NOSQL database tools.
12. Develop a simple GUI based database application and incorporate all the above-mentioned features
13. Case Study using any of the real life database applications from the following list

a) Inventory Management for a EMart Grocery Shop

b) Society Financial Management

c) Cop Friendly App – Eseva

d) Property Management – eMall

e) Star Small and Medium Banking and Finance

- Build Entity Model diagram. The diagram should align with the business and functional goals stated in the application.
- Apply Normalization rules in designing the tables in scope.
- Prepared applicable views, triggers (for auditing purposes), functions for enabling enterprise grade features.
- Build PL SQL / Stored Procedures for Complex Functionalities, ex EOD Batch Processing for calculating the EMI for Gold Loan for each eligible Customer.
- Ability to showcase ACID Properties with sample queries with appropriate settings

TOTAL: 45 PERIODS

COURSE OUTCOMES:

On completion of this course, the students will be able to:

CO1: Create databases with different types of key constraints

CO2: Construct simple and complex SQL queries using DML and DCL commands.

CO3: Use advanced features such as stored procedures and triggers and incorporate in GUI based application development.

CO4: Create an XML database and validate with meta-data (XML schema).

CO5: Create and manipulate data using NOSQL database.

LABORATORY REQUIREMENT

S.No	Equipment Name	Specification	Quantity
1.	Computer	AMD A6-7310 APU with AMD Radeon R4 Graphics 2.00 GHz, 4.00 GB RAM,64 bit Operating System , x64-based processor	For the batch of **30** students
2.	Software tools	SQL, MONGO DB, CASSENDRA, ORIENTDB, VISUAL STUDIO.NET	
3.	Platform	Windows 10	

EX.NO:1 **DATE :**	**DDL & DML COMMANDS**

AIM:

To create a database table, add constraints (primary key, unique, check, Not null), insert rows, update and delete rows using SQL DDL and DML commands.

DDL COMMANDS

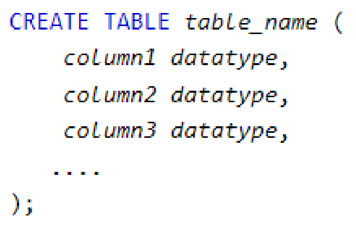
Data Definition Language Commands

DDL is short name of Data Definition Language, which deals with database schemas and descriptions, of how the datashould reside in the database.

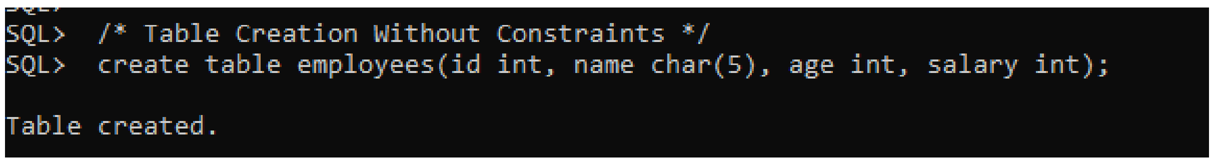
- CREATE - to create a database and its objects like (table, index, views, store procedure, function, and triggers)
- ALTER - alters the structure of the existing database
- DROP - delete objects from the database
- TRUNCATE - remove all records from a table, including all spaces allocated for the records are removed
- COMMENT - add comments to the data dictionary
- RENAME - rename an object

1.1 A) Create a Table without Constraints:

The CREATE TABLE statement is used to create a new table in a database. **SYNTAX**



```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
   ....
);
```

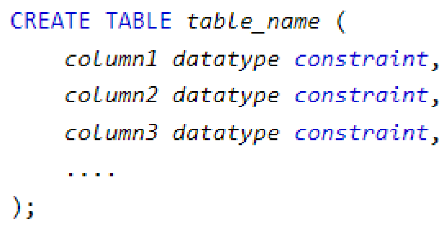


```
SQL>  /* Table Creation Without Constraints */
SQL>  create table employees(id int, name char(5), age int, salary int);

Table created.
```

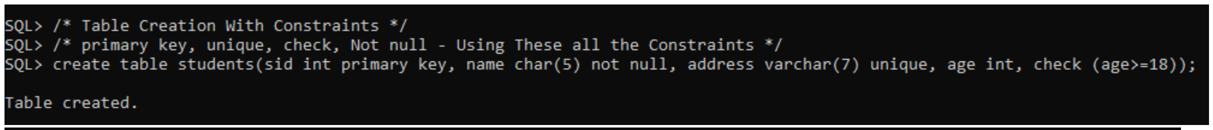
b) Add Constraints (Primary Key, Unique, Check, Not Null)

SQL constraints are used to specify rules for data in a table.

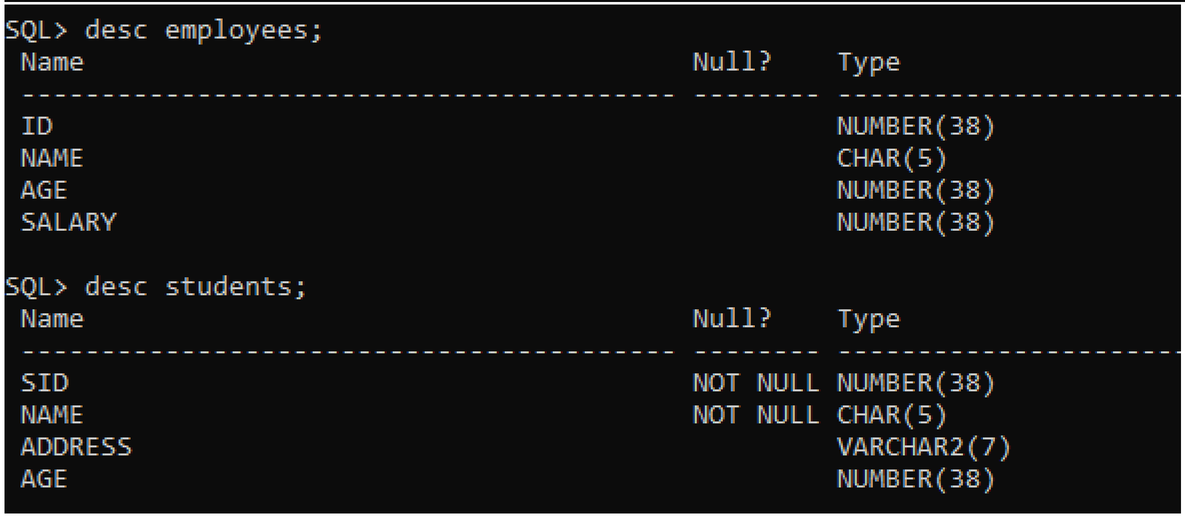
SYNTAX



```
CREATE TABLE table_name (
    column1 datatype constraint,
    column2 datatype constraint,
    column3 datatype constraint,
    ....
);
```



```
SQL> /* Table Creation With Constraints */
SQL> /* primary key, unique, check, Not null - Using These all the Constraints */
SQL> create table students(sid int primary key, name char(5) not null, address varchar(7) unique, age int, check (age>=18));

Table created.
```



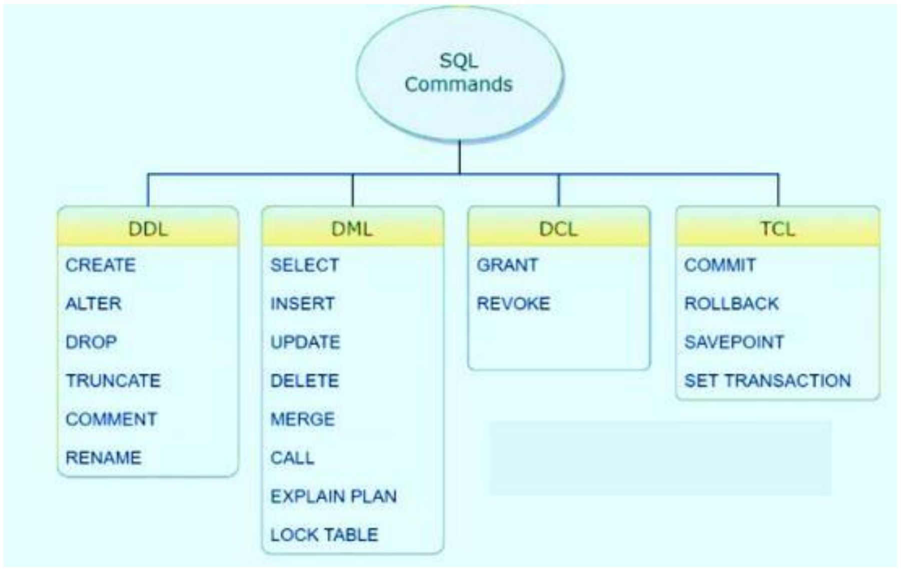
```
SQL> desc employees;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ID                                                 NUMBER(38)
 NAME                                               CHAR(5)
 AGE                                                NUMBER(38)
 SALARY                                             NUMBER(38)

SQL> desc students;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 SID                                       NOT NULL NUMBER(38)
 NAME                                      NOT NULL CHAR(5)
 ADDRESS                                            VARCHAR2(7)
 AGE                                                NUMBER(38)
```

Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

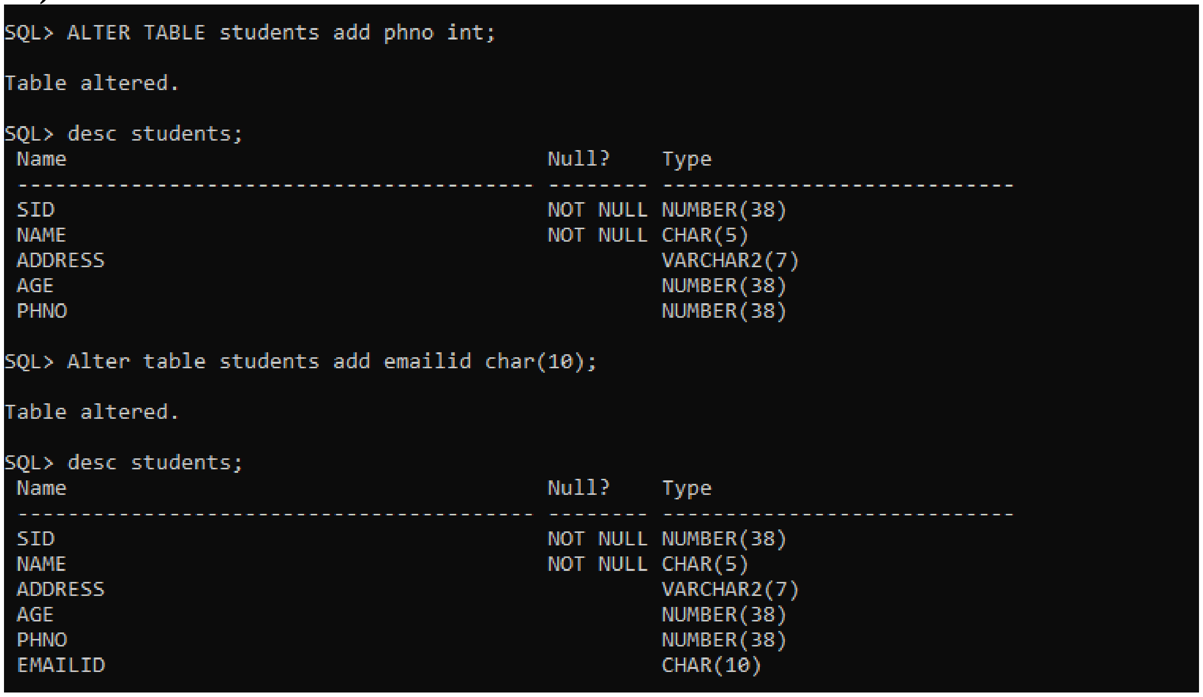
Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

The following constraints are commonly used in SQL:

- NOT NULL - Ensures that a column cannot have a NULL value
- UNIQUE - Ensures that all values in a column are different
- PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
- CHECK - Ensures that the values in a column satisfies a specific condition
- FOREIGN KEY - Prevents actions that would destroy links between tables
- DEFAULT - Sets a default value for a column if no value is specified
- CREATE INDEX - Used to create and retrieve data from the database very quick

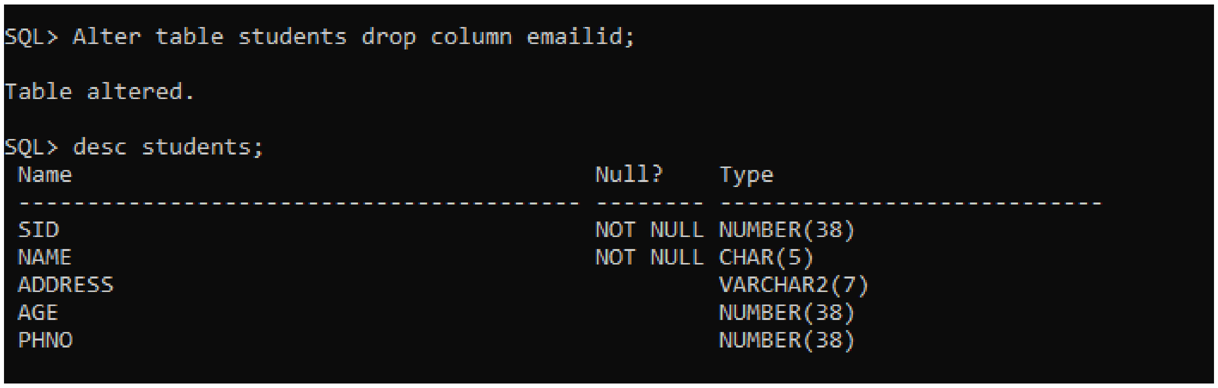


A) Alter a Table – Add Column



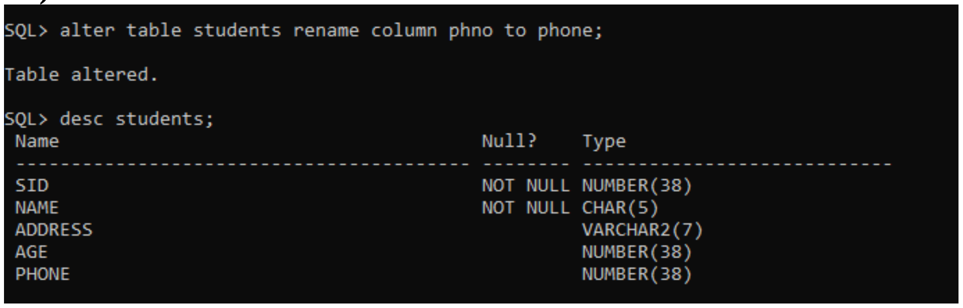
```
SQL> ALTER TABLE students add phno int;

Table altered.

SQL> desc students;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 SID                                       NOT NULL NUMBER(38)
 NAME                                      NOT NULL CHAR(5)
 ADDRESS                                            VARCHAR2(7)
 AGE                                                NUMBER(38)
 PHNO                                               NUMBER(38)

SQL> Alter table students add emailid char(10);

Table altered.

SQL> desc students;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 SID                                       NOT NULL NUMBER(38)
 NAME                                      NOT NULL CHAR(5)
 ADDRESS                                            VARCHAR2(7)
 AGE                                                NUMBER(38)
 PHNO                                               NUMBER(38)
 EMAILID                                            CHAR(10)
```

1.1.2 b) Alter a Table – Drop Column



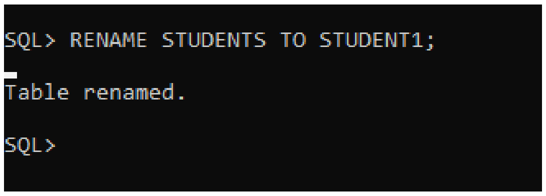
```
SQL> Alter table students drop column emailid;

Table altered.

SQL> desc students;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 SID                                       NOT NULL NUMBER(38)
 NAME                                      NOT NULL CHAR(5)
 ADDRESS                                            VARCHAR2(7)
 AGE                                                NUMBER(38)
 PHNO                                               NUMBER(38)
```

C) Alter a Table – Rename Column



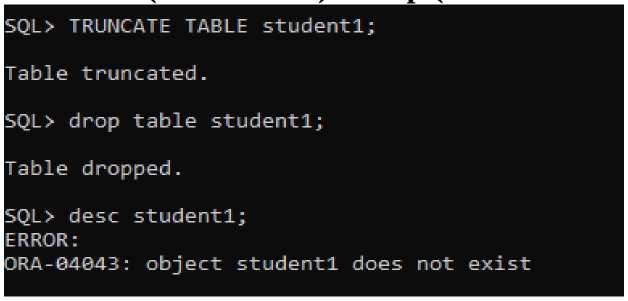
```
SQL> alter table students rename column phno to phone;

Table altered.

SQL> desc students;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 SID                                       NOT NULL NUMBER(38)
 NAME                                      NOT NULL CHAR(5)
 ADDRESS                                            VARCHAR2(7)
 AGE                                                NUMBER(38)
 PHONE                                              NUMBER(38)
```

Rename Table



```
SQL> RENAME STUDENTS TO STUDENT1;

Table renamed.

SQL>
```

Truncate (Clear Table) / Drop (Erase Whole Table);



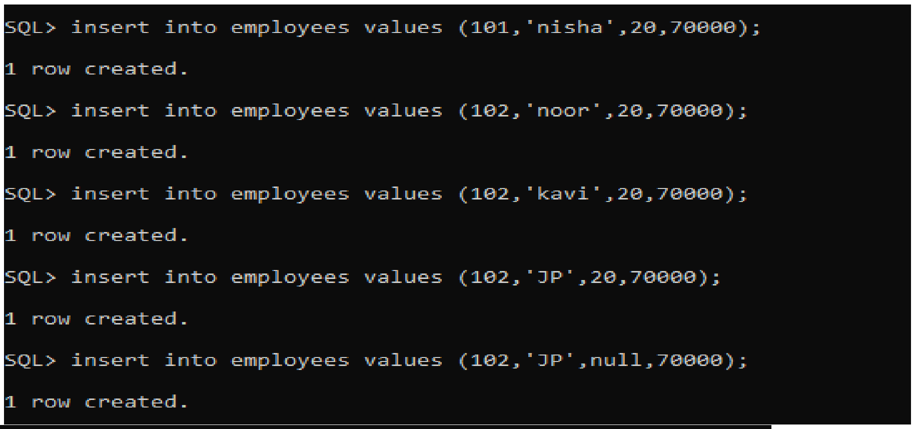
```
SQL> TRUNCATE TABLE student1;

Table truncated.

SQL> drop table student1;

Table dropped.

SQL> desc student1;
ERROR:
ORA-04043: object student1 does not exist
```

DATA MANIPULATION LANGUAGE COMMANDS

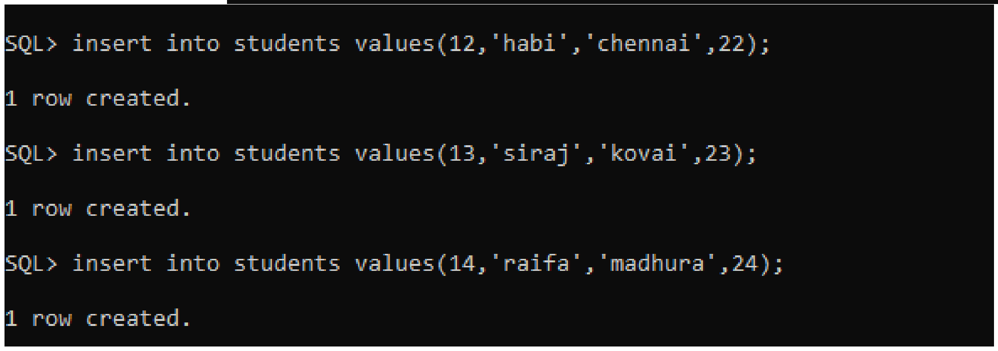
DML is short name of Data Manipulation Language which deals with data manipulation and includes most common SQL statements such SELECT, INSERT, UPDATE, DELETE, etc., and it is used to store, modify, retrieve, delete and update data in a database.

- INSERT - insert data into a table
- DELETE - Delete all records from a database table
- UPDATE - updates existing data within a table
- SELECT - retrieve data from a database

1.2.1 Insert Values into Tables:

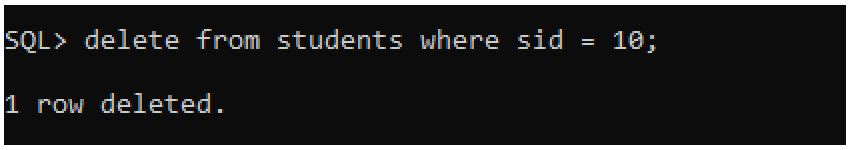


```
SQL> insert into employees values (101,'nisha',20,70000);

1 row created.

SQL> insert into employees values (102,'noor',20,70000);

1 row created.

SQL> insert into employees values (102,'kavi',20,70000);

1 row created.

SQL> insert into employees values (102,'JP',20,70000);

1 row created.

SQL> insert into employees values (102,'JP',null,70000);

1 row created.
```



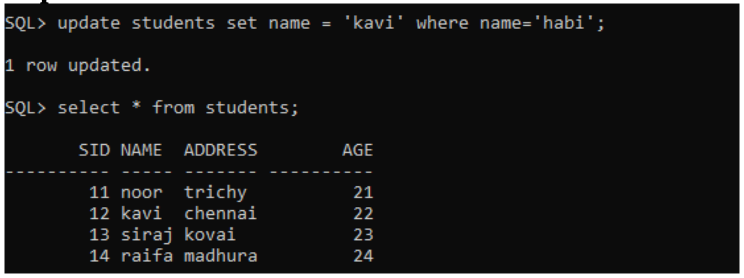
```
SQL> insert into students values(12,'habi','chennai',22);

1 row created.

SQL> insert into students values(13,'siraj','kovai',23);

1 row created.

SQL> insert into students values(14,'raifa','madhura',24);

1 row created.
```

Delete Values from Tables:



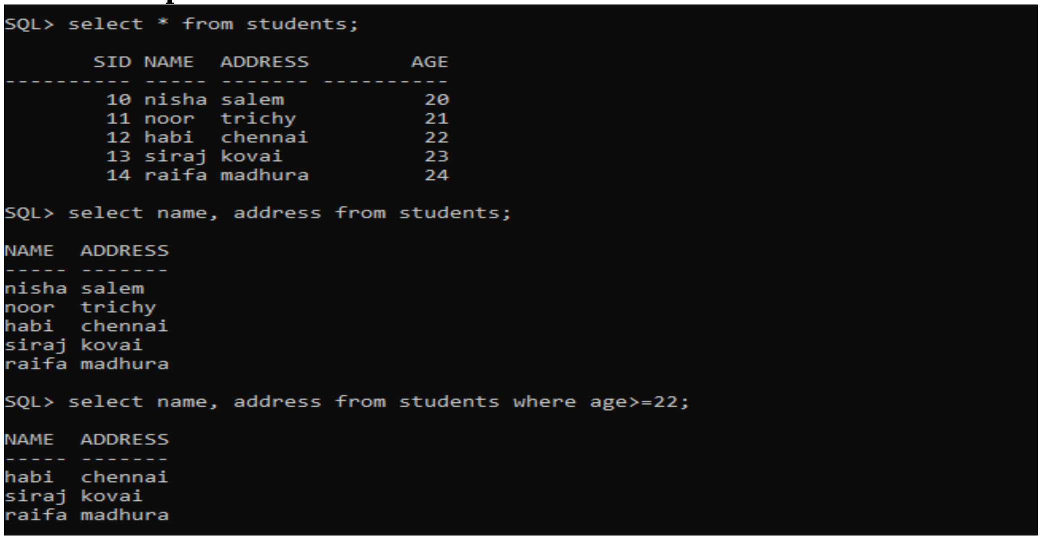
```
SQL> delete from students where sid = 10;

1 row deleted.
```

Update Values from the Table:

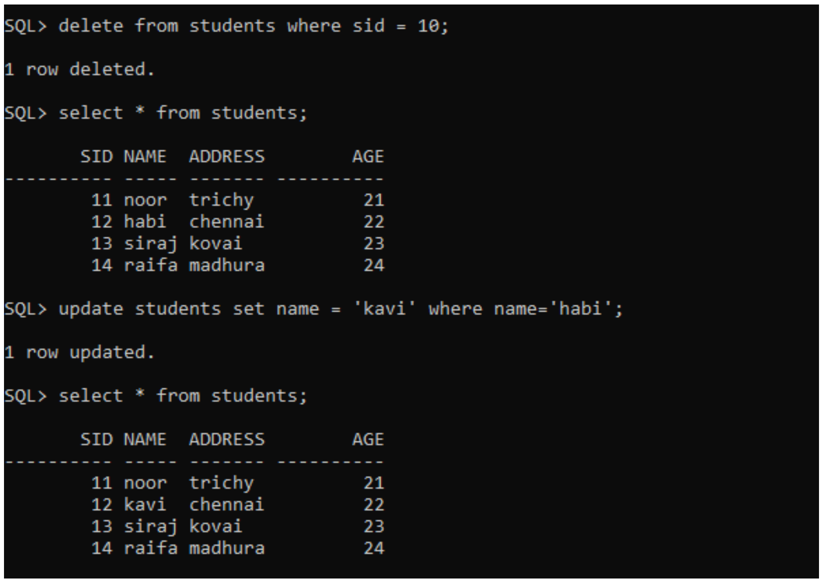


```
SQL> update students set name = 'kavi' where name='habi';

1 row updated.

SQL> select * from students;

       SID NAME  ADDRESS        AGE
---------- ----- ------- ----------
        11 noor  trichy          21
        12 kavi  chennai         22
        13 siraj kovai           23
        14 raifa madhura         24
```

SELECT operation from Table:



```
SQL> select * from students;

       SID NAME  ADDRESS         AGE
---------- ----- ------- ----------
        10 nisha salem            20
        11 noor  trichy           21
        12 habi  chennai          22
        13 siraj kovai            23
        14 raifa madhura          24

SQL> select name, address from students;

NAME  ADDRESS
----- -------
nisha salem
noor  trichy
habi  chennai
siraj kovai
raifa madhura

SQL> select name, address from students where age>=22;

NAME  ADDRESS
----- -------
habi  chennai
siraj kovai
raifa madhura
```



```
SQL> delete from students where sid = 10;

1 row deleted.

SQL> select * from students;

       SID NAME  ADDRESS         AGE
---------- ----- ------- ----------
        11 noor  trichy           21
        12 habi  chennai          22
        13 siraj kovai            23
        14 raifa madhura          24

SQL> update students set name = 'kavi' where name='habi';

1 row updated.

SQL> select * from students;

       SID NAME  ADDRESS         AGE
---------- ----- ------- ----------
        11 noor  trichy           21
        12 kavi  chennai          22
        13 siraj kovai            23
        14 raifa madhura          24
```

RESULT:

Thus the DDL, DML commands used to table was created with all constraints and executed successfully.

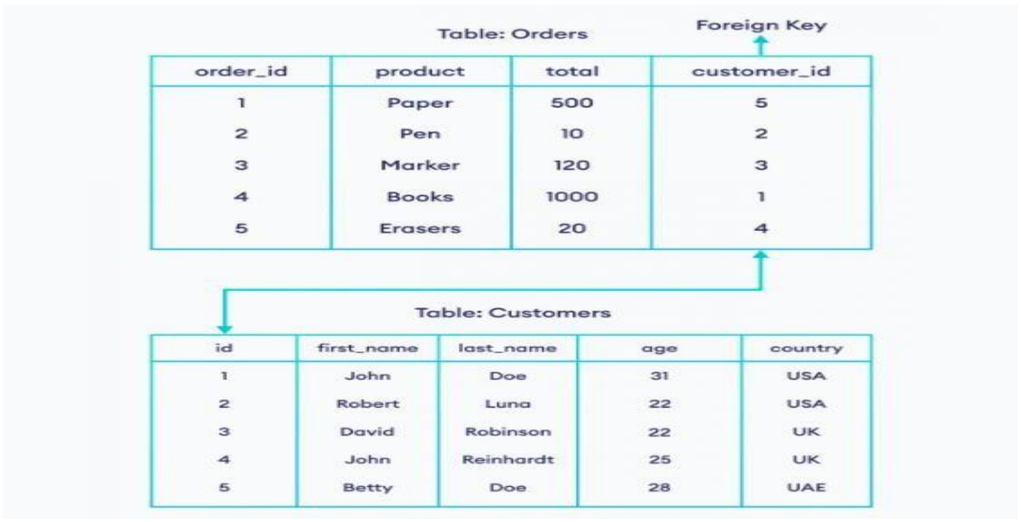
EX.NO:2 **DATE:**	**CONSTRAINTS**

AIM:

To create a set of tables, add foreign key constraints and incorporate referential integrity.

CONSTRAINTS:

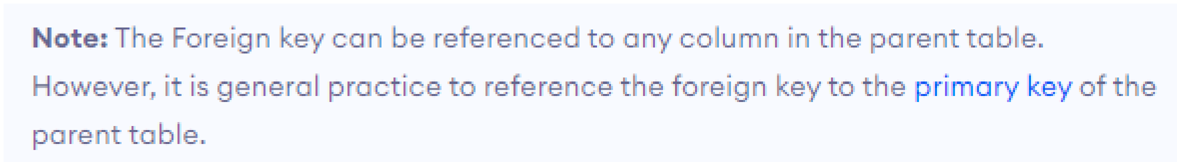
In SQL, we can create a relationship between two tables using the FOREIGN KEY constraint.



order_id	product	total	customer_id
1	Paper	500	5
2	Pen	10	2
3	Marker	120	3
4	Books	1000	1
5	Erasers	20	4

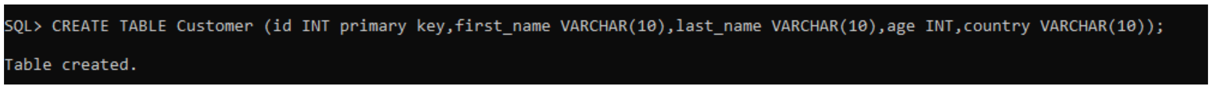
id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

Here, the customer_id field in the Orders table is FOREIGN KEY which references the id field in the Customers table. This means that the value of the customer_id (of the Orders table) must be a value from the id column (of the Customers table).

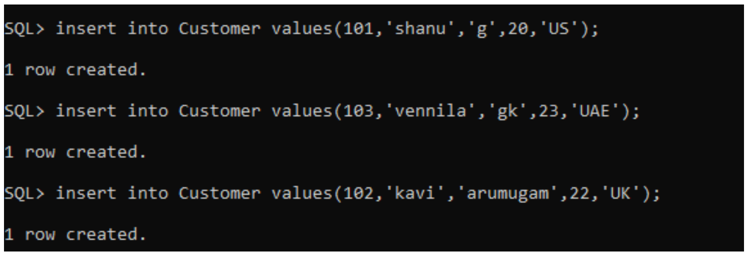


Note: The Foreign key can be referenced to any column in the parent table. However, it is general practice to reference the foreign key to the primary key of the parent table.

Without Foreign Key Table Creation:



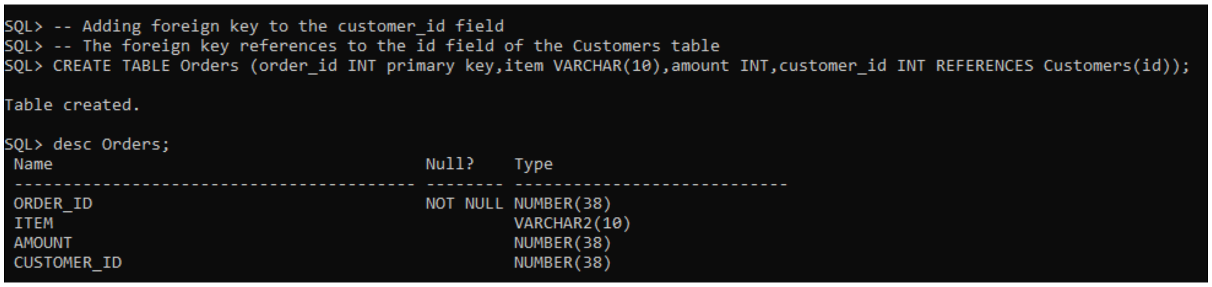
```
SQL> CREATE TABLE Customer (id INT primary key,first_name VARCHAR(10),last_name VARCHAR(10),age INT,country VARCHAR(10));

Table created.
```



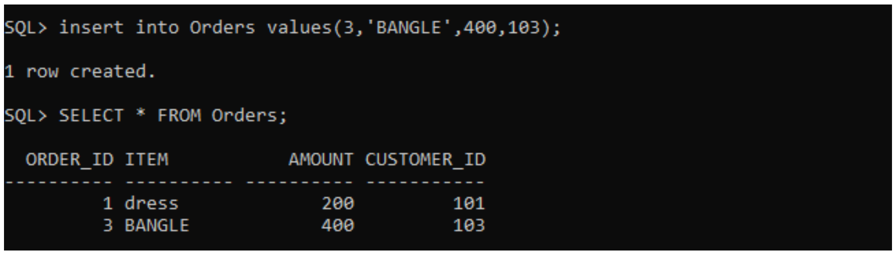
```
SQL> insert into Customer values(101,'shanu','g',20,'US');

1 row created.

SQL> insert into Customer values(103,'vennila','gk',23,'UAE');

1 row created.

SQL> insert into Customer values(102,'kavi','arumugam',22,'UK');

1 row created.
```

Creating FOREIGN Key

Now, let's see how we can create foreign key constraints in a database.

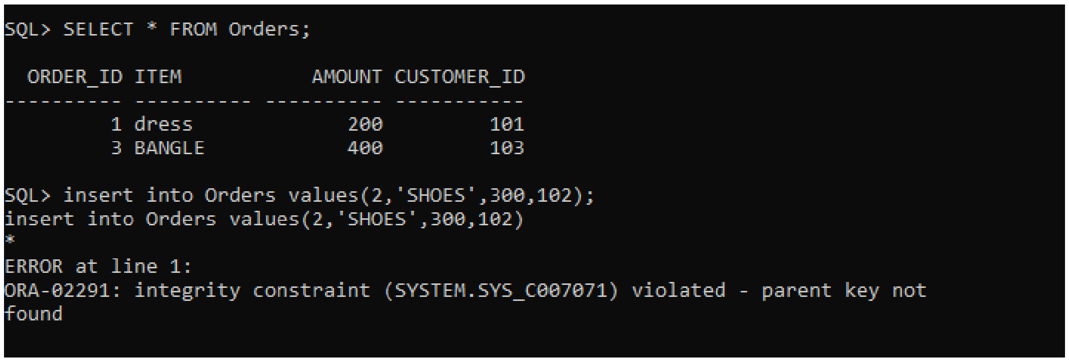


```
SQL> -- Adding foreign key to the customer_id field
SQL> -- The foreign key references to the id field of the Customers table
SQL> CREATE TABLE Orders (order_id INT primary key,item VARCHAR(10),amount INT,customer_id INT REFERENCES Customers(id));

Table created.

SQL> desc Orders;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ORDER_ID                                  NOT NULL NUMBER(38)
 ITEM                                               VARCHAR2(10)
 AMOUNT                                             NUMBER(38)
 CUSTOMER_ID                                        NUMBER(38)
```



```
SQL> insert into Orders values(3,'BANGLE',400,103);

1 row created.

SQL> SELECT * FROM Orders;

  ORDER_ID ITEM           AMOUNT CUSTOMER_ID
---------- ---------- ---------- -----------
         1 dress             200         101
         3 BANGLE            400         103
```

While Violating Foreign Key Rule



```
SQL> SELECT * FROM Orders;

  ORDER_ID ITEM           AMOUNT CUSTOMER_ID
---------- ---------- ---------- -----------
         1 dress             200         101
         3 BANGLE            400         103

SQL> insert into Orders values(2,'SHOES',300,102);
insert into Orders values(2,'SHOES',300,102)
*
ERROR at line 1:
ORA-02291: integrity constraint (SYSTEM.SYS_C007071) violated - parent key not
found
```

RESULT:

Thus the foreign key used to different set of table was created and executed successfully.

EX.NO:3 DATE:	**AGGREGATE FUNCTIONS**

AIM:

To query the database tables using different 'where' clause conditions and also implement aggregate functions.

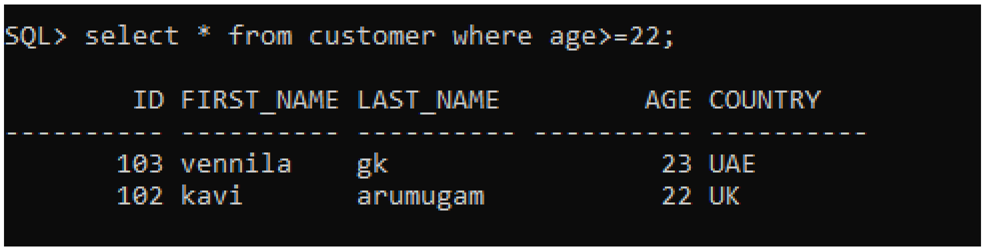
PROCEDURE:

The SQL WHERE clause is used to filter the results and apply conditions in a SELECT, INSERT, UPDATE, or DELETE statement. Syntax - The syntax for the WHERE clause in SQL is:

```
WHERE conditions;
```

One Condition in the WHERE Clause

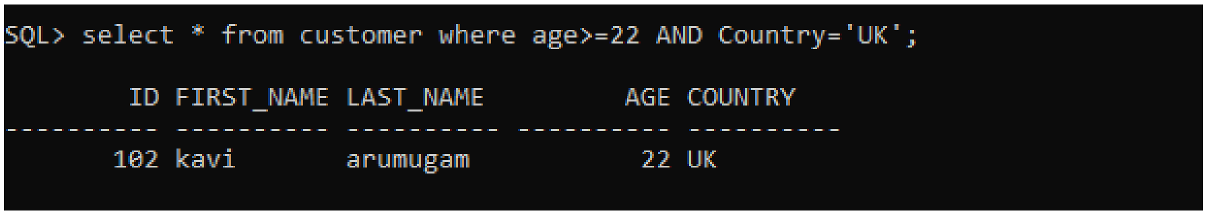
It is difficult to explain the syntax for the SQL WHERE clause, so let's start with an example that uses the WHERE clause to apply 1 condition.



```
SQL> select * from customer where age>=22;

        ID FIRST_NAME LAST_NAME         AGE COUNTRY
---------- ---------- ---------- ---------- ----------
       103 vennila    gk                 23 UAE
       102 kavi       arumugam           22 UK
```

Two Conditions in the WHERE Clause (AND Condition)

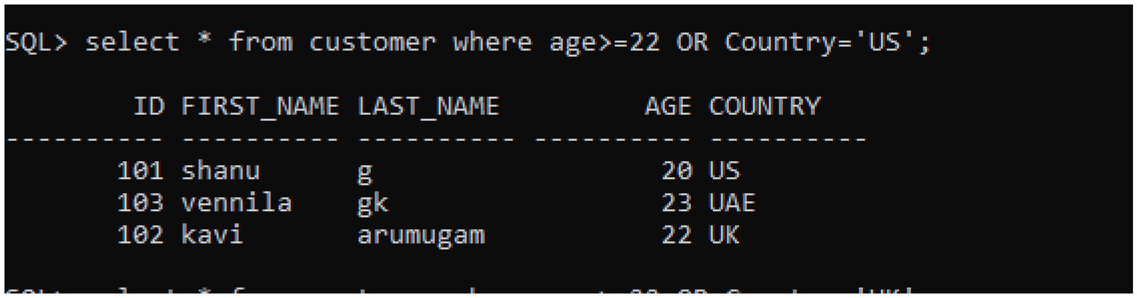
You can use the AND condition in the WHERE clause to specify more than 1 condition that must be met for the record to be selected. Let's explore how to do this.



```
SQL> select * from customer where age>=22 AND Country='UK';

        ID FIRST_NAME LAST_NAME         AGE COUNTRY
---------- ---------- ---------- ---------- ----------
       102 kavi       arumugam           22 UK
```

Two Conditions in the WHERE Clause (OR Condition)

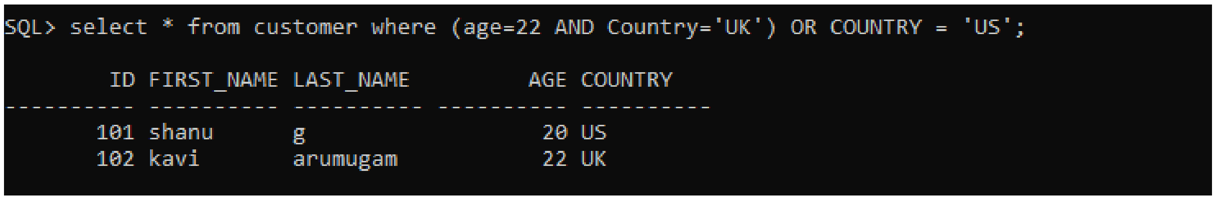
You can use the OR condition in the WHERE clause to test multiple conditions where the record is returned if any one of the conditions are met.



```
SQL> select * from customer where age>=22 OR Country='US';

        ID FIRST_NAME LAST_NAME         AGE COUNTRY
---------- ---------- ---------- ---------- ----------
       101 shanu      g                  20 US
       103 vennila    gk                 23 UAE
       102 kavi       arumugam           22 UK
```

Combining AND & OR conditions

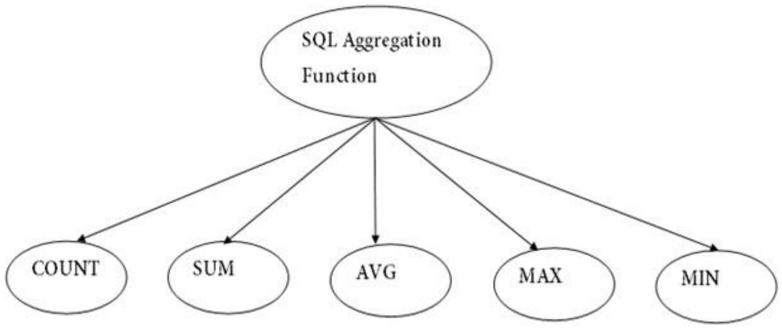
You can also combine the AND condition with the OR condition to test more complex conditions.



```
SQL> select * from customer where (age=22 AND Country='UK') OR COUNTRY = 'US';

        ID FIRST_NAME LAST_NAME         AGE COUNTRY
---------- ---------- ---------- ---------- ----------
       101 shanu      g                  20 US
       102 kavi       arumugam           22 UK
```

Aggregate function

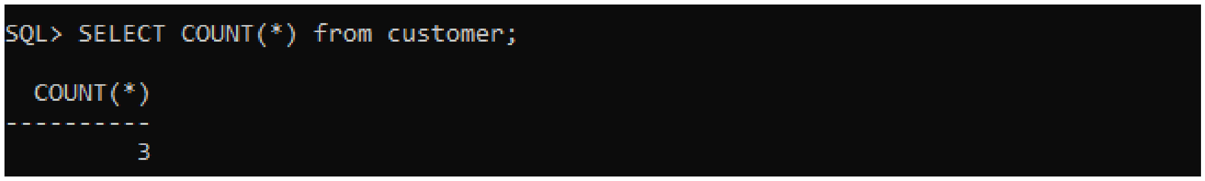
SQL aggregation function is used to perform the calculations on multiple rows of a single column of a table. It returns asingle value. It is also used to summarize the data.



COUNT FUNCTION

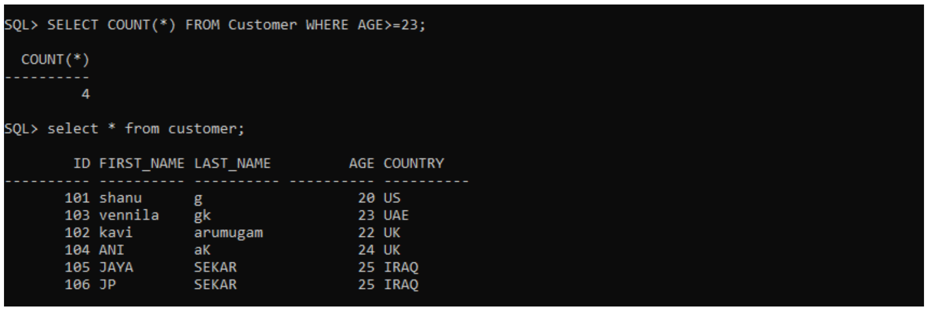
COUNT function is used to Count the number of rows in a database table. It can work on both numeric and non-numericdata types.

COUNT function uses the COUNT(*) that returns the count of all the rows in a specified table. COUNT(*) considersduplicate and Null.



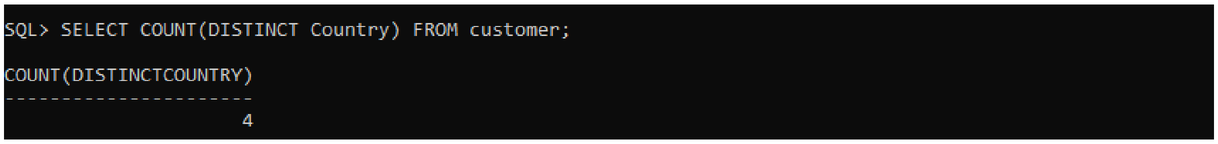
```
SQL> SELECT COUNT(*) from customer;

  COUNT(*)
----------
         3
```

COUNT with WHERE



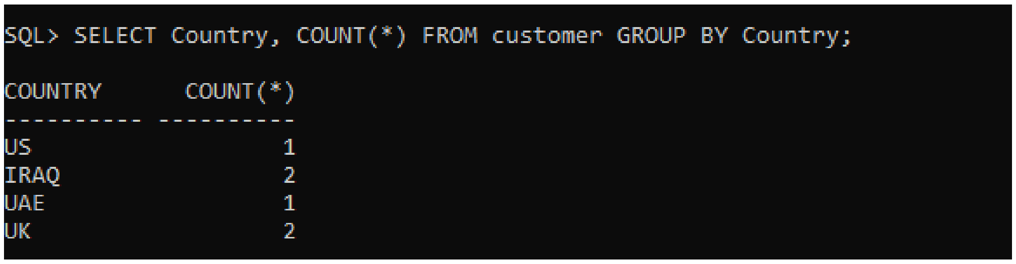
```
SQL> SELECT COUNT(*) FROM Customer WHERE AGE>=23;

  COUNT(*)
----------
         4

SQL> select * from customer;

        ID FIRST_NAME LAST_NAME         AGE COUNTRY
---------- ---------- ---------- ---------- ----------
       101 shanu      g                  20 US
       103 vennila    gk                 23 UAE
       102 kavi       arumugam           22 UK
       104 ANI        aK                 24 UK
       105 JAYA       SEKAR              25 IRAQ
       106 JP         SEKAR              25 IRAQ
```

COUNT() with DISTINCT



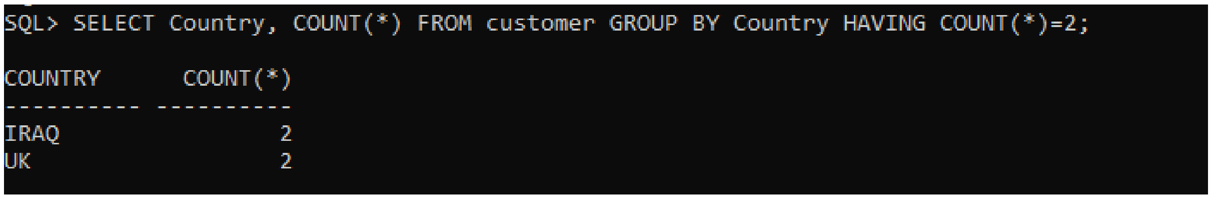
```
SQL> SELECT COUNT(DISTINCT Country) FROM customer;

COUNT(DISTINCTCOUNTRY)
----------------------
                     4
```

COUNT() with GROUP BY



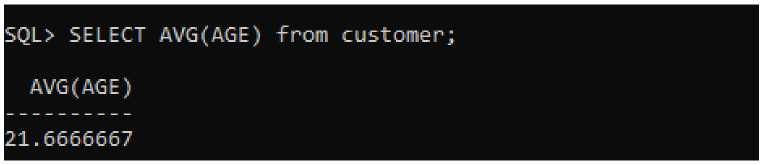
```
SQL> SELECT Country, COUNT(*) FROM customer GROUP BY Country;

COUNTRY      COUNT(*)
---------- ----------
US                  1
IRAQ                2
UAE                 1
UK                  2
```

COUNT() with HAVING



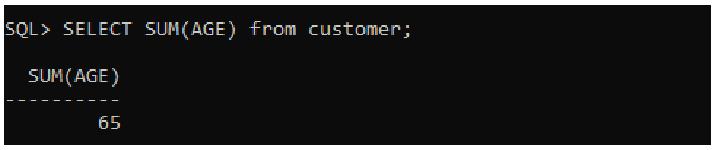
```
SQL> SELECT Country, COUNT(*) FROM customer GROUP BY Country HAVING COUNT(*)=2;

COUNTRY      COUNT(*)
---------- ----------
IRAQ                2
UK                  2
```

Average Function



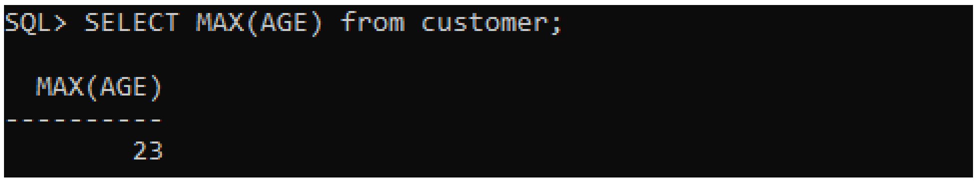
```
SQL> SELECT AVG(AGE) from customer;

  AVG(AGE)
----------
21.6666667
```

SUM Function

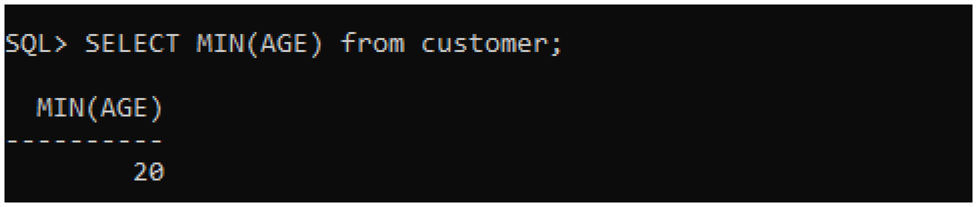


```
SQL> SELECT SUM(AGE) from customer;

  SUM(AGE)
----------
        65
```

MAX Function


```
SQL> SELECT MAX(AGE) from customer;

  MAX(AGE)
----------
        23
```

MIN Function


```
SQL> SELECT MIN(AGE) from customer;

  MIN(AGE)
----------
        20
```

RESULT:

Thus the database tables used to different 'where' clause conditions applied to various aggregate functions are executedsuccessfully.

EX.NO:4 DATE:	SUB QUERIES AND SIMPLE JOIN OPERATIONS

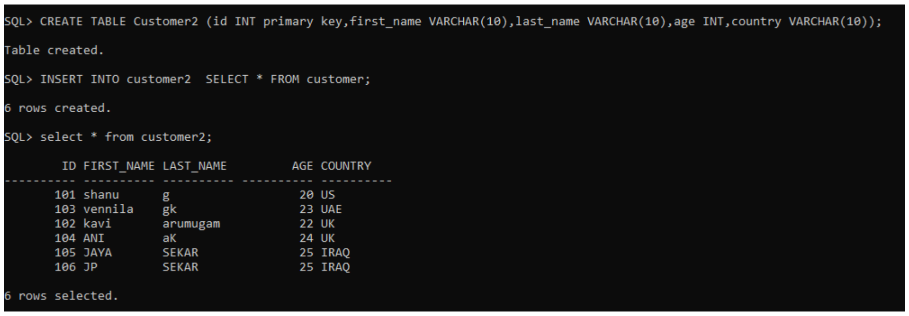
AIM:

To create Database table and explore various sub queries using insert, delete, update and select command and alsoperform join operations.

PROCEDURE:

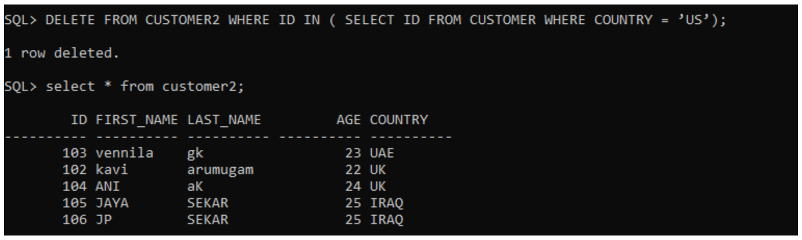
- In SQL a Sub query can be simply defined as a query within another query. In other words we can say that a Sub query is a query that is embedded in WHERE clause of another SQL query. Important rules for Sub queries:
- You can place the Sub query in a number of SQL clauses: WHERE clause, HAVING clause, FROM clause. Sub queriescan be used with SELECT, UPDATE, INSERT, DELETE statements along with expression operator. It could be equalityoperator or comparison operator such as =, >, =, <= and Like operator.
- A sub query is a query within another query. The outer query is called as main query and inner query is called as sub query.
- The sub query generally executes first when the sub query doesn't have any co-relation with the main query, whenthere is a co-relation the parser takes the decision on the fly on which query to execute on precedence and uses the output of the sub query accordingly. Sub query must be enclosed in parentheses. Sub queries are on the rightside of the comparison operator.
- ORDER BY command cannot be used in a Sub query. GROUPBY command can be used to perform same function as ORDER BY command.
- Use single-row operators with single row Sub queries. Use multiple-row operators with multiple-row Sub queries.

Sub Queries – (Create Duplicate Table – USING INSERT & SELECT COMMANDS)



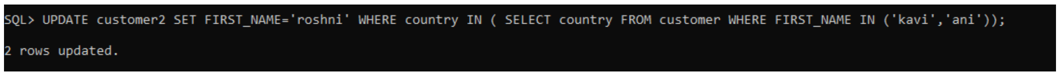
```
SQL> CREATE TABLE Customer2 (id INT primary key,first_name VARCHAR(10),last_name VARCHAR(10),age INT,country VARCHAR(10));

Table created.

SQL> INSERT INTO customer2  SELECT * FROM customer;

6 rows created.

SQL> select * from customer2;

        ID FIRST_NAME LAST_NAME         AGE COUNTRY
---------- ---------- ---------- ---------- ----------
       101 shanu      g                  20 US
       103 vennila    gk                 23 UAE
       102 kavi       arumugam           22 UK
       104 ANI        aK                 24 UK
       105 JAYA       SEKAR              25 IRAQ
       106 JP         SEKAR              25 IRAQ

6 rows selected.
```

Sub Queries – (DELETE COMMANDS)

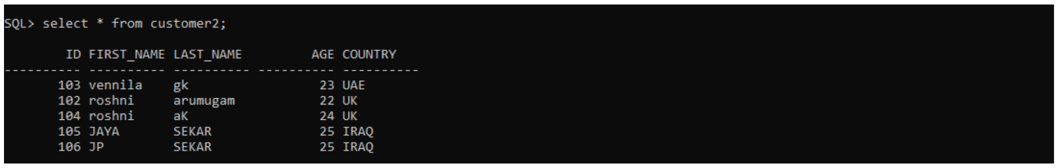


```
SQL> DELETE FROM CUSTOMER2 WHERE ID IN ( SELECT ID FROM CUSTOMER WHERE COUNTRY = 'US');

1 row deleted.

SQL> select * from customer2;

        ID FIRST_NAME LAST_NAME         AGE COUNTRY
---------- ---------- ---------- ---------- ----------
       103 vennila    gk                 23 UAE
       102 kavi       arumugam           22 UK
       104 ANI        aK                 24 UK
       105 JAYA       SEKAR              25 IRAQ
       106 JP         SEKAR              25 IRAQ
```

Sub Queries – (UPDATE COMMANDS)

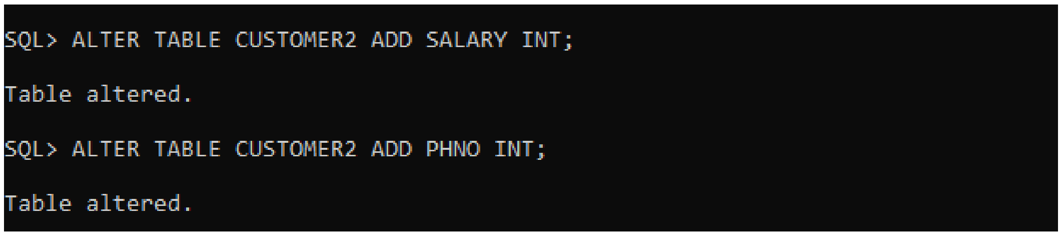


```
SQL> UPDATE customer2 SET FIRST_NAME='roshni' WHERE country IN ( SELECT country FROM customer WHERE FIRST_NAME IN ('kavi','ani'));

2 rows updated.
```

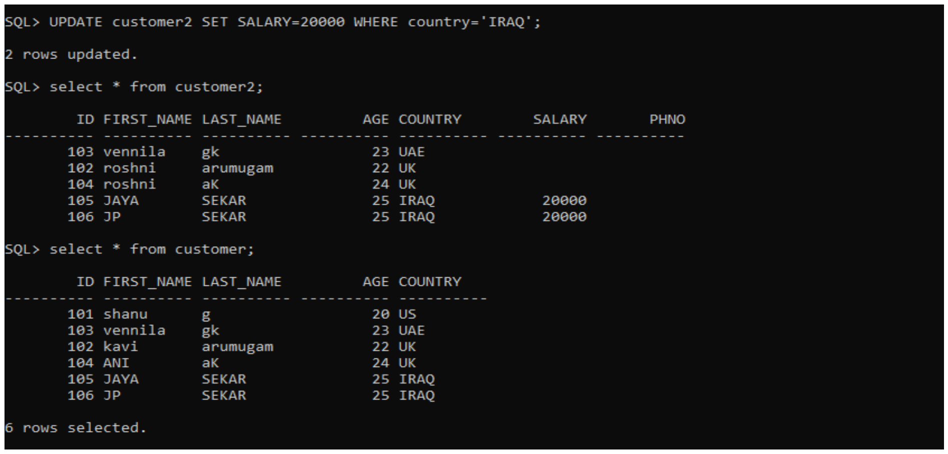


```
SQL> select * from customer2;

        ID FIRST_NAME LAST_NAME         AGE COUNTRY
---------- ---------- ---------- ---------- ----------
       103 vennila    gk                 23 UAE
       102 roshni     arumugam           22 UK
       104 roshni     aK                 24 UK
       105 JAYA       SEKAR              25 IRAQ
       106 JP         SEKAR              25 IRAQ
```

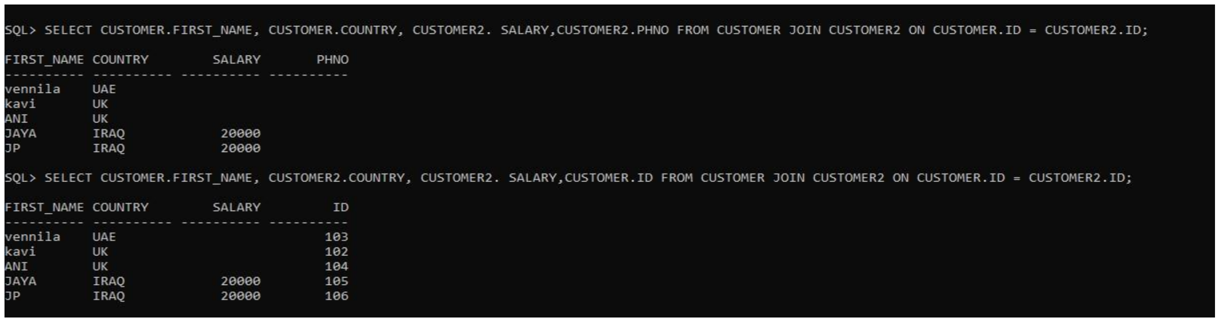
SUB QURIES – SIMPLE JOIN



```
SQL> ALTER TABLE CUSTOMER2 ADD SALARY INT;

Table altered.

SQL> ALTER TABLE CUSTOMER2 ADD PHNO INT;

Table altered.
```



```
SQL> UPDATE customer2 SET SALARY=20000 WHERE country='IRAQ';

2 rows updated.

SQL> select * from customer2;

        ID FIRST_NAME LAST_NAME         AGE COUNTRY        SALARY       PHNO
---------- ---------- ---------- ---------- ---------- ---------- ----------
       103 vennila    gk                 23 UAE
       102 roshni     arumugam           22 UK
       104 roshni     aK                 24 UK
       105 JAYA       SEKAR              25 IRAQ            20000
       106 JP         SEKAR              25 IRAQ            20000

SQL> select * from customer;

        ID FIRST_NAME LAST_NAME         AGE COUNTRY
---------- ---------- ---------- ---------- ----------
       101 shanu      g                  20 US
       103 vennila    gk                 23 UAE
       102 kavi       arumugam           22 UK
       104 ANI        aK                 24 UK
       105 JAYA       SEKAR              25 IRAQ
       106 JP         SEKAR              25 IRAQ

6 rows selected.
```



```
SQL> SELECT CUSTOMER.FIRST_NAME, CUSTOMER.COUNTRY, CUSTOMER2. SALARY,CUSTOMER2.PHNO FROM CUSTOMER JOIN CUSTOMER2 ON CUSTOMER.ID = CUSTOMER2.ID;

FIRST_NAME COUNTRY        SALARY       PHNO
---------- ---------- ---------- ----------
vennila    UAE
kavi       UK
ANI        UK
JAYA       IRAQ            20000
JP         IRAQ            20000

SQL> SELECT CUSTOMER.FIRST_NAME, CUSTOMER2.COUNTRY, CUSTOMER2. SALARY,CUSTOMER.ID FROM CUSTOMER JOIN CUSTOMER2 ON CUSTOMER.ID = CUSTOMER2.ID;

FIRST_NAME COUNTRY        SALARY         ID
---------- ---------- ---------- ----------
vennila    UAE                          103
kavi       UK                           102
ANI        UK                           104
JAYA       IRAQ            20000        105
JP         IRAQ            20000        106
```

RESULT:

Thus the SQL Program used to Database table was created and explored various sub queries using insert, delete, update and select command and also performed join operations successfully.

EX.NO:5 **DATE:**	**NATURAL, EQUI AND OUTER JOINS**

AIM:

To query the database tables and explore natural, equi and outer joins.

PROCEDURE:

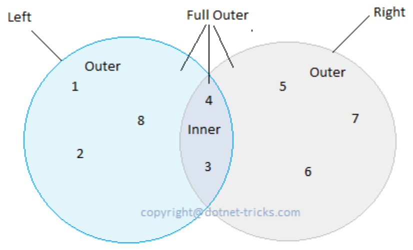
SQL Joins are used to fetch/retrieve data from two or more data tables, based on a join condition. A join condition is a relationship among some columns in the data tables that take part in SQL join. Basically, database tables are related to each other with keys. We use this keys relationship in SQL Joins.

Types of SQL Joins

In SQL Server we have only three types of joins. Using these joins we fetch the data from multiple tables based on condition.

Inner Join / EQUI Join

Inner join returns only those records/rows that match/exist in both the tables. The inner join generally depends upon FORM or the WHERE clause in which the data of the first table is joined using another table using the terms 'inner join' followed by the second table to be joined with the first table.



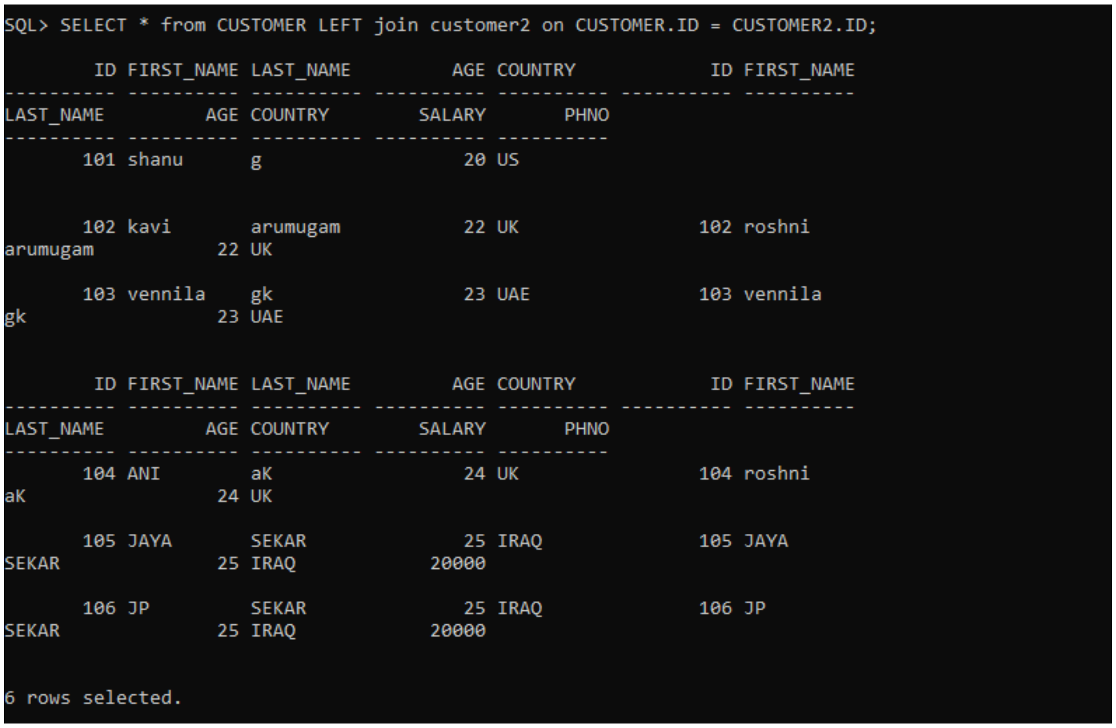
Inner Join Result : (4,3)
Left Join Result : (1,2,8,4,3)
Right Join Result : (5,6,7,4,3)
Full Outer Join Result : (1,2,8,4,3,5,6,7)

Outer Join

Outer join is also called Right join and the primary reason a right join would be used is when we are joining more than twotables from the database. In these use-cases, using a right join method is preferable because it can avoid restructuring ourwhole query to join one table. Outside of this thing, the right joins are used very rarely due to their complexity, so for such Simple joins, it’s better to use a left join than a right as it will be easier for our query to be read and understood by others while developing a DBMS query. We have three types of Outer Join.

Left Outer Join

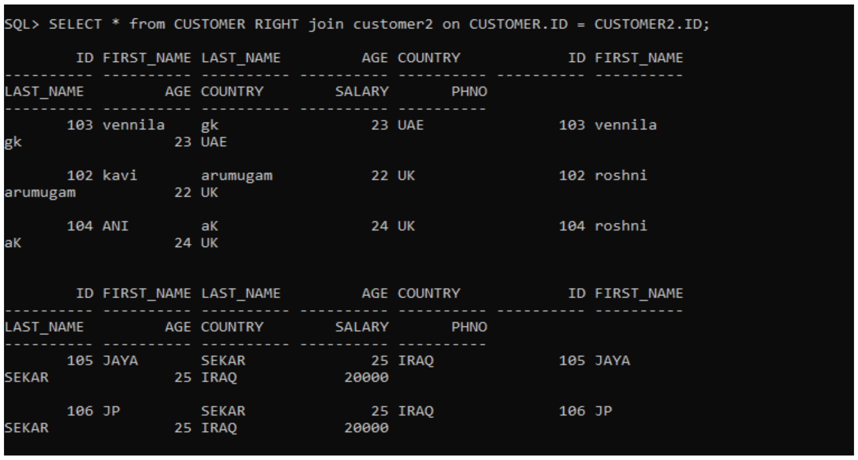
Left outer join returns all records/rows from the left table and from the right table returns only matched records. If thereare no columns matching in the right table, it returns NULL values.



```
SQL> SELECT * from CUSTOMER LEFT join customer2 on CUSTOMER.ID = CUSTOMER2.ID;

        ID FIRST_NAME LAST_NAME         AGE COUNTRY            ID FIRST_NAME
---------- ---------- ---------- ---------- ---------- ---------- ----------
LAST_NAME         AGE COUNTRY        SALARY       PHNO
---------- ---------- ---------- ---------- ----------
       101 shanu      g                 20 US


       102 kavi       arumugam          22 UK                102 roshni
arumugam          22 UK

       103 vennila    gk                23 UAE               103 vennila
gk                23 UAE


        ID FIRST_NAME LAST_NAME         AGE COUNTRY            ID FIRST_NAME
---------- ---------- ---------- ---------- ---------- ---------- ----------
LAST_NAME         AGE COUNTRY        SALARY       PHNO
---------- ---------- ---------- ---------- ----------
       104 ANI        aK                24 UK                104 roshni
aK                24 UK

       105 JAYA       SEKAR             25 IRAQ              105 JAYA
SEKAR             25 IRAQ            20000

       106 JP         SEKAR             25 IRAQ              106 JP
SEKAR             25 IRAQ            20000


6 rows selected.
```

Right Outer Join

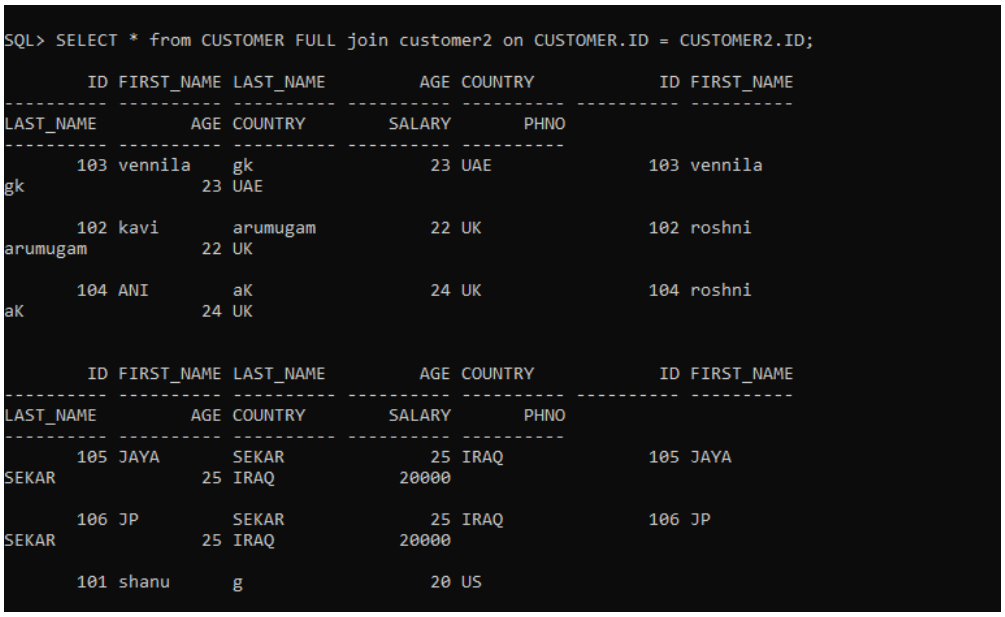
A right outer join returns all records/rows from the right table and from the left table returns only matched records. Ifthere are no columns matching in the left table, it returns NULL values.



```
SQL> SELECT * from CUSTOMER RIGHT join customer2 on CUSTOMER.ID = CUSTOMER2.ID;

        ID FIRST_NAME LAST_NAME         AGE COUNTRY            ID FIRST_NAME
---------- ---------- ---------- ---------- ---------- ---------- ----------
LAST_NAME         AGE COUNTRY        SALARY       PHNO
---------- ---------- ---------- ---------- ----------
       103 vennila    gk                23 UAE               103 vennila
gk                23 UAE

       102 kavi       arumugam          22 UK                102 roshni
arumugam          22 UK

       104 ANI        aK                24 UK                104 roshni
aK                24 UK


        ID FIRST_NAME LAST_NAME         AGE COUNTRY            ID FIRST_NAME
---------- ---------- ---------- ---------- ---------- ---------- ----------
LAST_NAME         AGE COUNTRY        SALARY       PHNO
---------- ---------- ---------- ---------- ----------
       105 JAYA       SEKAR             25 IRAQ              105 JAYA
SEKAR             25 IRAQ            20000

       106 JP         SEKAR             25 IRAQ              106 JP
SEKAR             25 IRAQ            20000
```

Full Outer Join

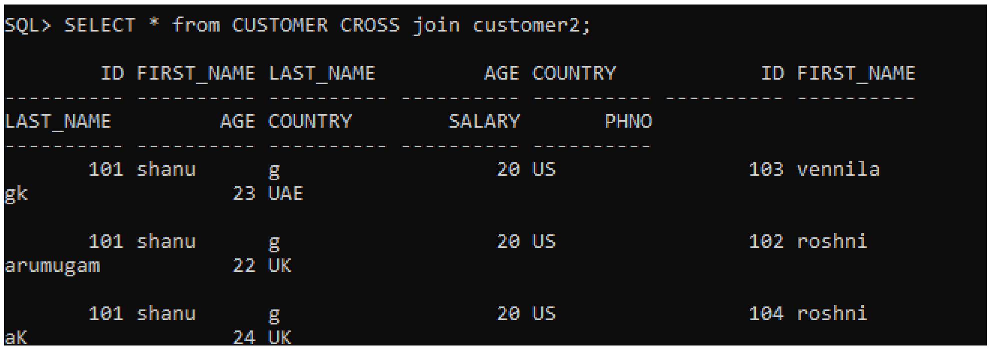
Full outer join combines left outer join and right outer join. This join returns all records/rows from both tables. If there areno columns matching in both tables, it returns NULL values.



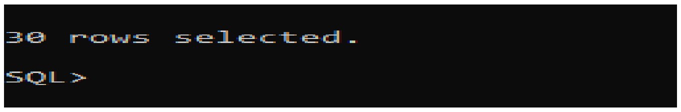
```
SQL> SELECT * from CUSTOMER FULL join customer2 on CUSTOMER.ID = CUSTOMER2.ID;

        ID FIRST_NAME LAST_NAME         AGE COUNTRY            ID FIRST_NAME
---------- ---------- ---------- ---------- ---------- ---------- ----------
LAST_NAME         AGE COUNTRY        SALARY       PHNO
---------- ---------- ---------- ---------- ----------
       103 vennila    gk                 23 UAE               103 vennila
gk                 23 UAE

       102 kavi       arumugam           22 UK                102 roshni
arumugam           22 UK

       104 ANI        aK                 24 UK                104 roshni
aK                 24 UK


        ID FIRST_NAME LAST_NAME         AGE COUNTRY            ID FIRST_NAME
---------- ---------- ---------- ---------- ---------- ---------- ----------
LAST_NAME         AGE COUNTRY        SALARY       PHNO
---------- ---------- ---------- ---------- ----------
       105 JAYA       SEKAR              25 IRAQ              105 JAYA
SEKAR              25 IRAQ            20000

       106 JP         SEKAR              25 IRAQ              106 JP
SEKAR              25 IRAQ            20000

       101 shanu      g                  20 US
```

Cross Join

Cross join is a cartesian join means the cartesian product of both the tables. This join does not need any condition to join two tables. This join returns records/rows that are multiplication of record numbers from both the tables means each rowon the left table will be related to each row of the right table.



```
SQL> SELECT * from CUSTOMER CROSS join customer2;

        ID FIRST_NAME LAST_NAME         AGE COUNTRY            ID FIRST_NAME
---------- ---------- ---------- ---------- ---------- ---------- ----------
LAST_NAME         AGE COUNTRY        SALARY       PHNO
---------- ---------- ---------- ---------- ----------
       101 shanu      g                  20 US                103 vennila
gk                 23 UAE

       101 shanu      g                  20 US                102 roshni
arumugam           22 UK

       101 shanu      g                  20 US                104 roshni
aK                 24 UK
```



```
30 rows selected.

SQL>
```

Self Join

Self-join is used to join a database table to itself, particularly when the table has a foreign key that references its own Primary Key. Basically, we have only three types of joins: Inner join, Outer join, and Cross join. We use any of these three JOINS to join a table to itself. Hence Self-join is not a type of SQL join.

Syntax for ALL TYPES OF JOINS

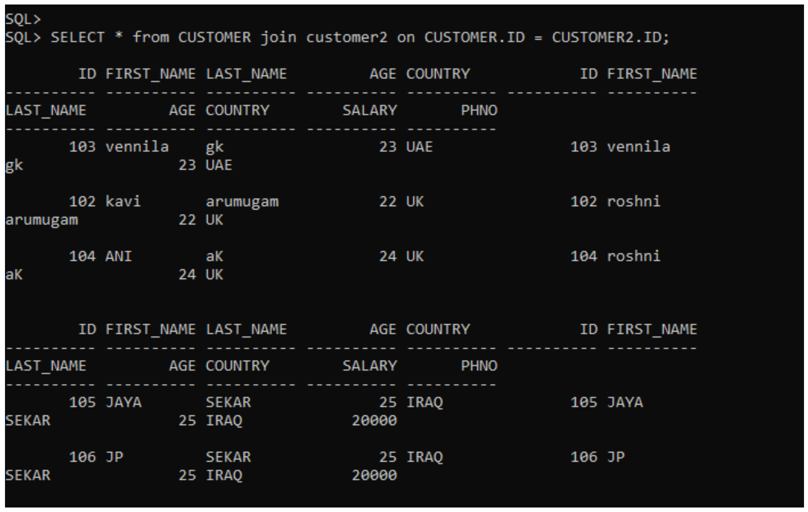
```
Select * from table_1 as t1
inner join table_2 as t2
on t1.IDcol=t2.IDcol
```

```
Select * from table_1 as t1
left outer join table_2 as t2
on t1.IDcol=t2.IDcol
```

```
Select * from table_1 as t1
right outer join table_2 as t2
on t1.IDcol=t2.IDcol
```

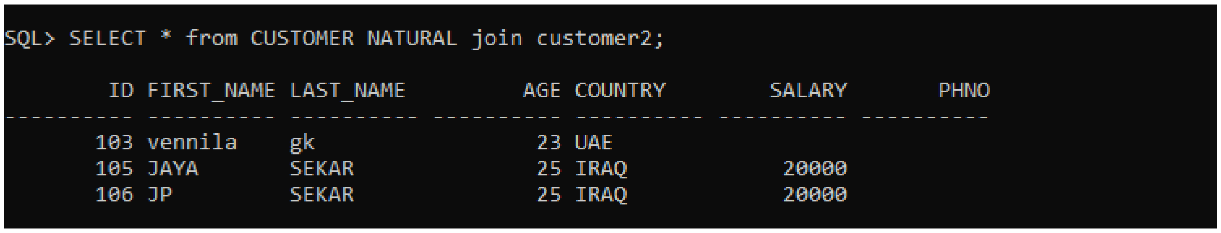
```
Select * from table_1 as t1
full outer join table_2 as t2
on t1.IDcol=t2.IDcol
```

```
Select * from table_1
cross join table_2
```



```
SQL>
SQL> SELECT * from CUSTOMER join customer2 on CUSTOMER.ID = CUSTOMER2.ID;

        ID FIRST_NAME LAST_NAME         AGE COUNTRY            ID FIRST_NAME
---------- ---------- ---------- ---------- ---------- ---------- ----------
LAST_NAME         AGE COUNTRY        SALARY       PHNO
---------- ---------- ---------- ---------- ----------
       103 vennila    gk                 23 UAE               103 vennila
gk                 23 UAE

       102 kavi       arumugam           22 UK                102 roshni
arumugam           22 UK

       104 ANI        aK                 24 UK                104 roshni
aK                 24 UK


        ID FIRST_NAME LAST_NAME         AGE COUNTRY            ID FIRST_NAME
---------- ---------- ---------- ---------- ---------- ---------- ----------
LAST_NAME         AGE COUNTRY        SALARY       PHNO
---------- ---------- ---------- ---------- ----------
       105 JAYA       SEKAR              25 IRAQ              105 JAYA
SEKAR              25 IRAQ            20000

       106 JP         SEKAR              25 IRAQ              106 JP
SEKAR              25 IRAQ            20000
```

Natural Join



```
SQL> SELECT * from CUSTOMER NATURAL join customer2;

        ID FIRST_NAME LAST_NAME         AGE COUNTRY        SALARY       PHNO
---------- ---------- ---------- ---------- ---------- ---------- ----------
       103 vennila    gk                 23 UAE
       105 JAYA       SEKAR              25 IRAQ            20000
       106 JP         SEKAR              25 IRAQ            20000
```

RESULT:

Thus the SQL Program used to Database table was created and also various join queries operations was executedsuccessfully.

EX.NO:6 **DATE:**	**FUNCTIONS & PROCEDURES**

AIM:

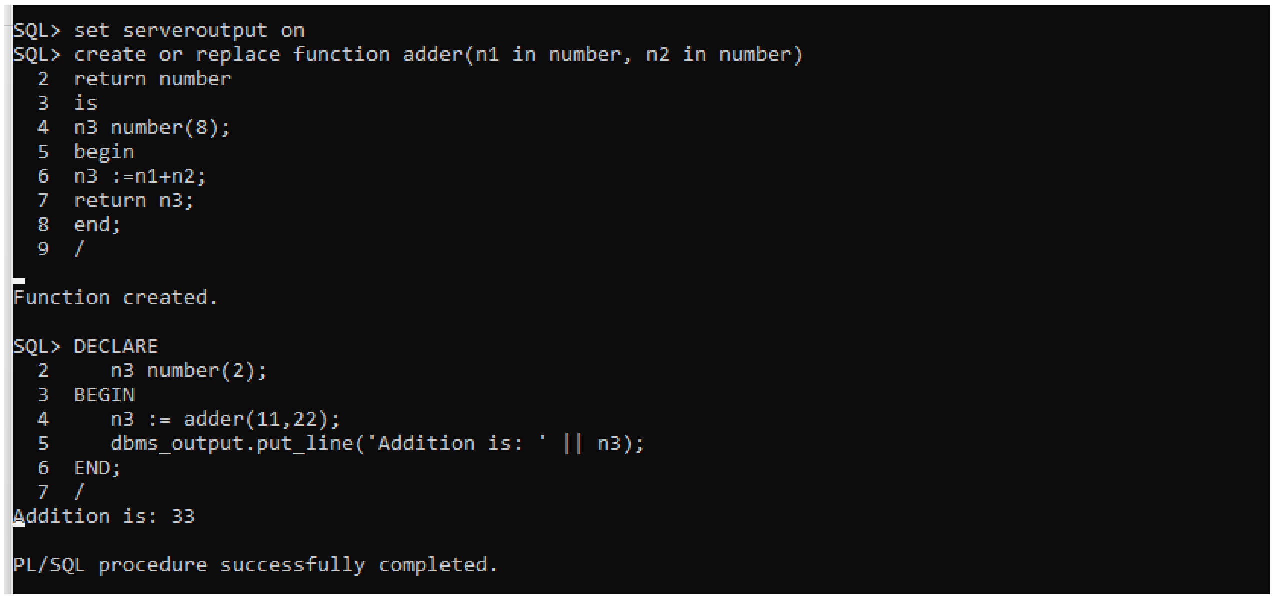
To create a user defined function and stored procedure using SQL.

PROCEDURE:

The PL/SQL Function is very similar to PL/SQL Procedure. The main difference between procedure and a function is, a function must always return a value, and on the other hand a procedure may or may not return a value. Except this, all the other things of PL/SQL procedure are true for PL/SQL function too

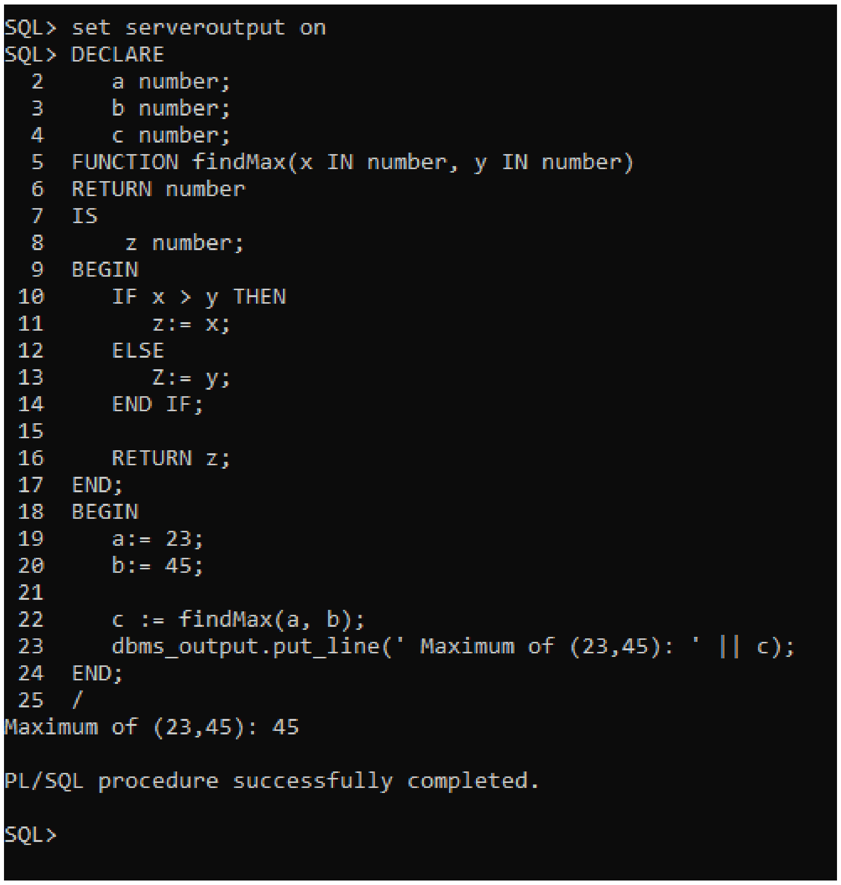
```
CREATE [OR REPLACE] FUNCTION function_name [parameters]
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
RETURN return_datatype
{IS | AS}
BEGIN
  < function_body >
END [function_name];
```

Addition of TWO Numbers:



```
SQL> set serveroutput on
SQL> create or replace function adder(n1 in number, n2 in number)
return number
is
n3 number(8);
begin
n3 :=n1+n2;
return n3;
end;
/

Function created.

SQL> DECLARE
   n3 number(2);
BEGIN
   n3 := adder(11,22);
   dbms_output.put_line('Addition is: ' || n3);
END;
/
Addition is: 33

PL/SQL procedure successfully completed.
```

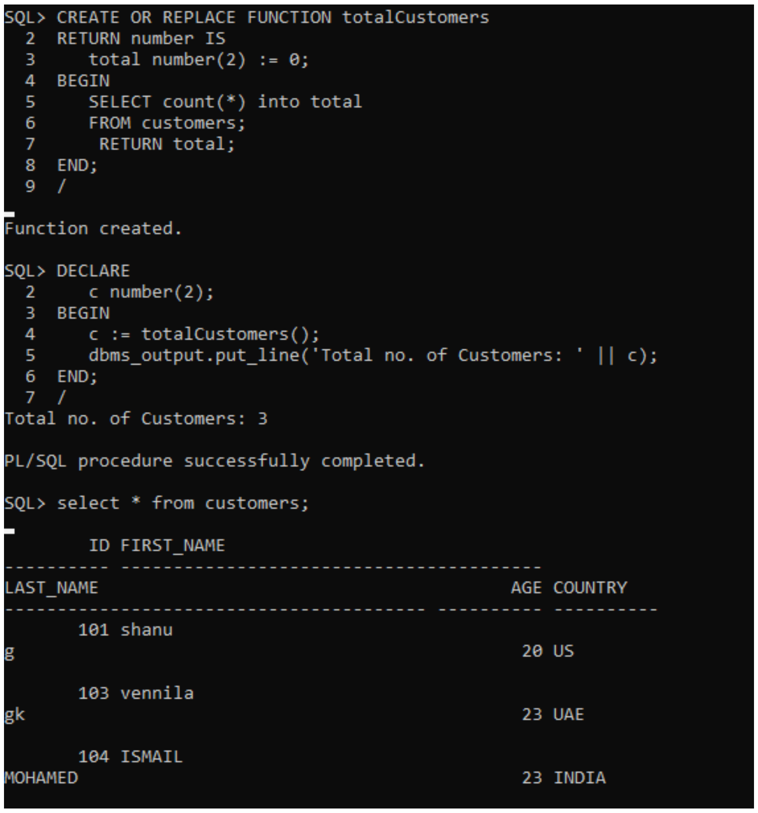
Maximum of TWO Numbers



```
SQL> set serveroutput on
SQL> DECLARE
  2     a number;
  3     b number;
  4     c number;
  5  FUNCTION findMax(x IN number, y IN number)
  6  RETURN number
  7  IS
  8      z number;
  9  BEGIN
 10     IF x > y THEN
 11        z:= x;
 12     ELSE
 13        Z:= y;
 14     END IF;
 15
 16     RETURN z;
 17  END;
 18  BEGIN
 19     a:= 23;
 20     b:= 45;
 21
 22     c := findMax(a, b);
 23     dbms_output.put_line(' Maximum of (23,45): ' || c);
 24  END;
 25  /
Maximum of (23,45): 45

PL/SQL procedure successfully completed.

SQL>
```

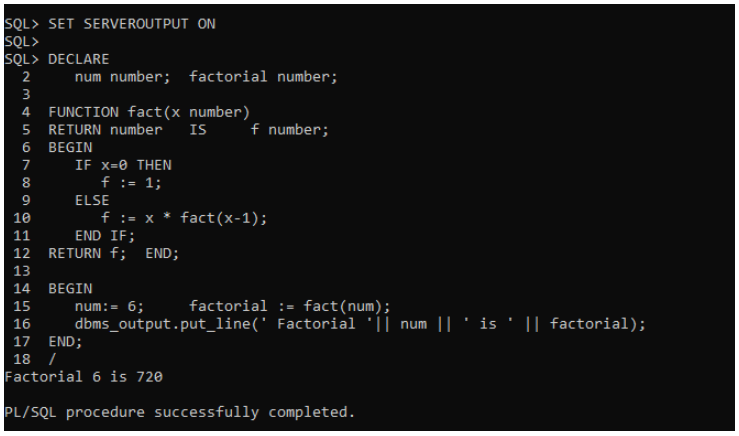
Calling PL/SQL Function:

While creating a function, you have to give a definition of what the function has to do. To use a function, you will have to call that function to perform the defined task. Once the function is called, the program control is transferred to the calledfunction. After the successful completion of the defined task, the call function returns program control back to the main program.



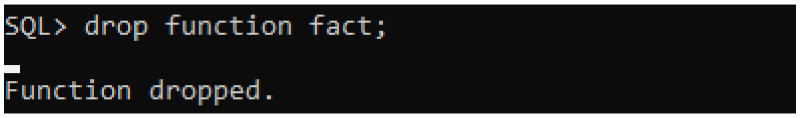
```
SQL> CREATE OR REPLACE FUNCTION totalCustomers
RETURN number IS
   total number(2) := 0;
BEGIN
   SELECT count(*) into total
   FROM customers;
    RETURN total;
END;
/

Function created.

SQL> DECLARE
   c number(2);
BEGIN
   c := totalCustomers();
   dbms_output.put_line('Total no. of Customers: ' || c);
END;
/
Total no. of Customers: 3

PL/SQL procedure successfully completed.

SQL> select * from customers;

        ID FIRST_NAME
---------- ----------------------------------------
LAST_NAME                                       AGE COUNTRY
---------------------------------------- ---------- ----------
       101 shanu
g                                                20 US

       103 vennila
gk                                               23 UAE

       104 ISMAIL
MOHAMED                                          23 INDIA
```

PL/SQL Recursive Function



```
SQL> SET SERVEROUTPUT ON
SQL>
SQL> 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;
/
Factorial 6 is 720

PL/SQL procedure successfully completed.
```

Drop Function



```
SQL> drop function fact;

Function dropped.
```

RESULT:

Thus the PL/SQL functions program was created and executed successfully.

EX.NO:7 DATE:	DCL & TCL COMMANDS

AIM

To execute complex transactions and also realize various DCL and TCL commands.

PROCEDURE:

Data Control Language Commands

DCL is short name of Data Control Language which includes commands such as GRANT and mostly concerned with rights, permissions and other controls of the database system.

- GRANT - allow users access privileges to the database
- REVOKE - withdraw users access privileges given by using the GRANT command

Queries:

Tables Used: Consider the following tables namely "DEPARTMENTS" and"EMPLOYEES"

Their schemas are as follows , Departments (dept _no , dept_ name , dept_location);Employees (emp_id , emp_name , emp_salary);

Q1: Develop a query to grant all privileges of employees table into departments table

Ans: SQL> Grant all on employees to departments;

Grant succeeded.

Q2: Develop a query to grant some privileges of employees table into departments table

Ans: SQL> Grant select, update , insert on departments to departments with grant option;Grant succeeded.

Q3: Develop a query to revoke all privileges of employees table from departments table

Ans: SQL> Revoke all on employees from departments; Revoke succeeded.

Q4: Develop a query to revoke some privileges of employees table from departments table

Ans: SQL> Revoke select, update , insert on departments from departments;

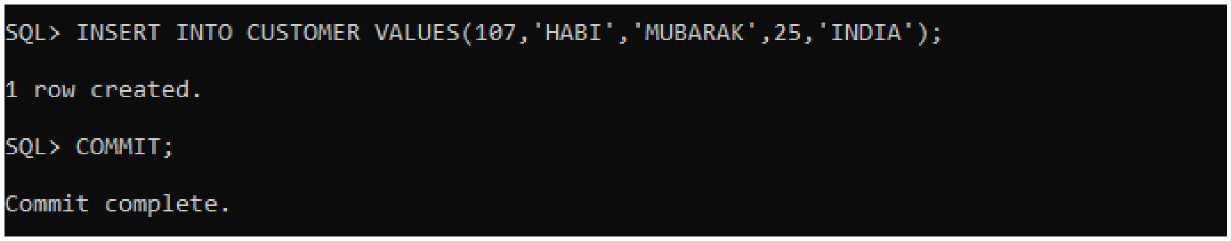
Revoke succeeded.

Transaction Control Language Commands:

TCL is short name of Transaction Control Language which deals with a transaction within a database.

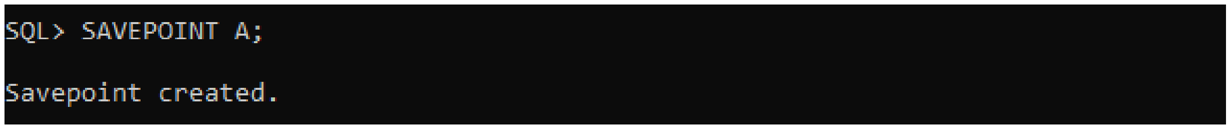
- COMMIT - commits a Transaction
- SAVEPOINT - to roll back the transaction making points within groups
- ROLLBACK - rollback a transaction in case of any error occurs

COMMIT



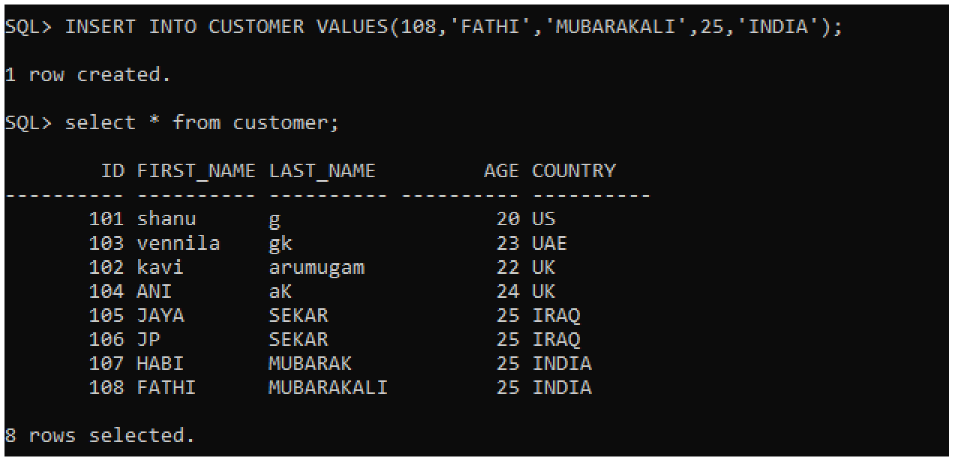
```
SQL> INSERT INTO CUSTOMER VALUES(107,'HABI','MUBARAK',25,'INDIA');

1 row created.

SQL> COMMIT;

Commit complete.
```

SAVE POINT

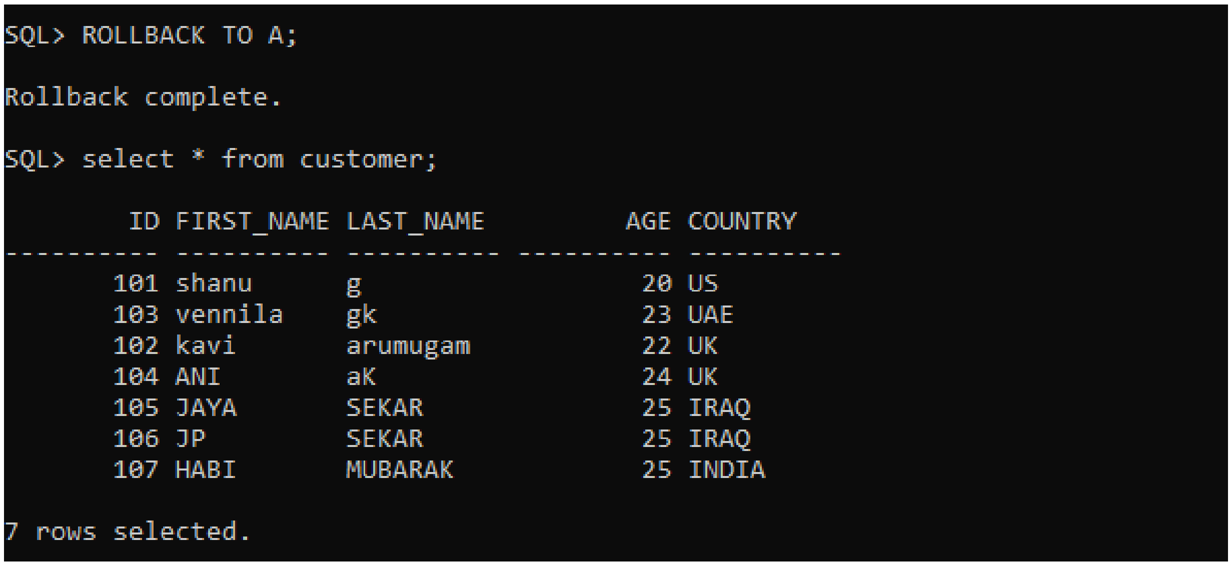


```
SQL> SAVEPOINT A;

Savepoint created.
```

ROLLBACK



```
SQL> INSERT INTO CUSTOMER VALUES(108,'FATHI','MUBARAKALI',25,'INDIA');

1 row created.

SQL> select * from customer;

        ID FIRST_NAME LAST_NAME         AGE COUNTRY
---------- ---------- ---------- ---------- ----------
       101 shanu      g                  20 US
       103 vennila    gk                 23 UAE
       102 kavi       arumugam           22 UK
       104 ANI        aK                 24 UK
       105 JAYA       SEKAR              25 IRAQ
       106 JP         SEKAR              25 IRAQ
       107 HABI       MUBARAK            25 INDIA
       108 FATHI      MUBARAKALI         25 INDIA

8 rows selected.
```



```
SQL> ROLLBACK TO A;

Rollback complete.

SQL> select * from customer;

        ID FIRST_NAME LAST_NAME         AGE COUNTRY
---------- ---------- ---------- ---------- ----------
       101 shanu      g                  20 US
       103 vennila    gk                 23 UAE
       102 kavi       arumugam           22 UK
       104 ANI        aK                 24 UK
       105 JAYA       SEKAR              25 IRAQ
       106 JP         SEKAR              25 IRAQ
       107 HABI       MUBARAK            25 INDIA

7 rows selected.
```

RESULT:
Thus the SQL program used to complex transactions DCL and TCL commands are executed successfully.

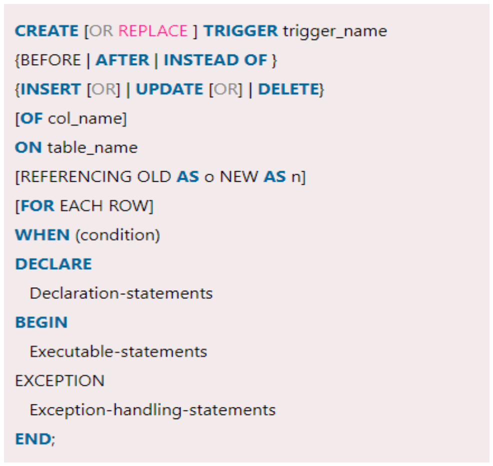
EX.NO:8 DATE:	SQL TRIGGERS

AIM:

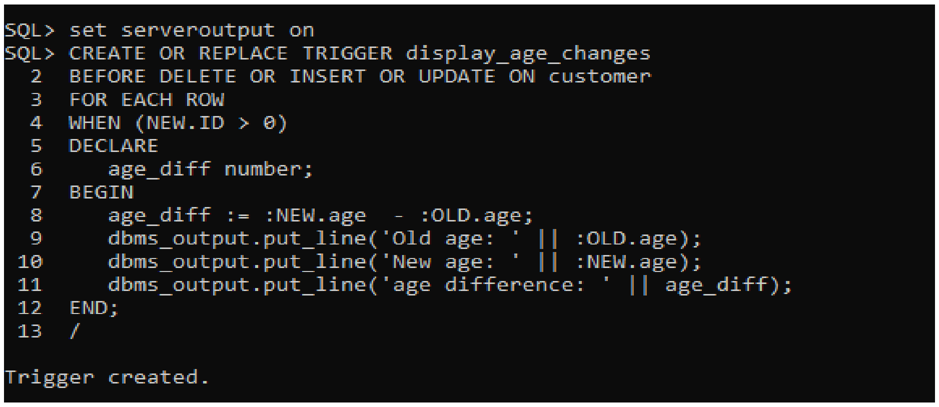
To write SQL Triggers for insert, delete, and update operations in a database table.

PROCEDURE:

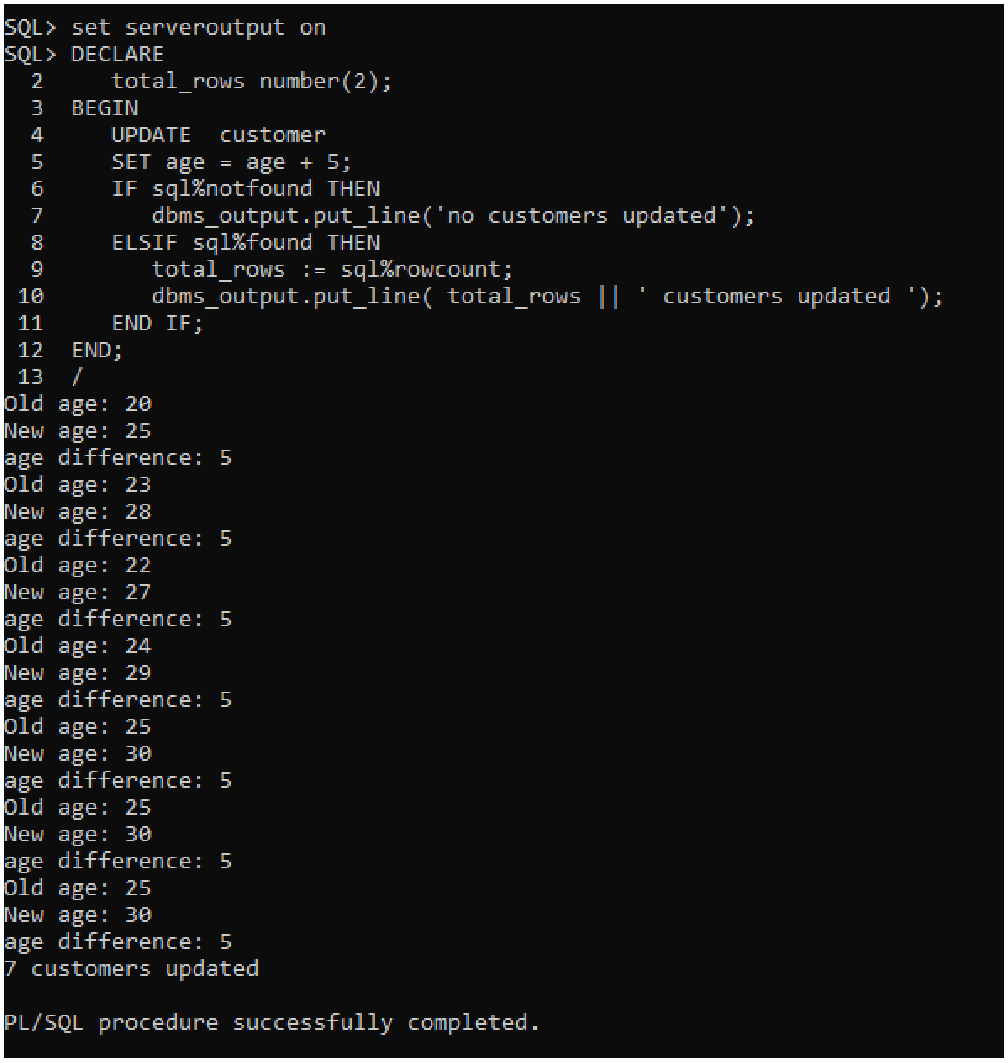
- Trigger is invoked by Oracle engine automatically whenever a specified event occurs. Trigger is stored intodatabase and invoked repeatedly, when specific condition match.
- Triggers are stored programs, which are automatically executed or fired when some event occurs.
- Triggers are written to be executed in response to any of the following events.
- A database manipulation (DML) statement (DELETE, INSERT, or UPDATE).
- A database definition (DDL) statement (CREATE, ALTER, or DROP).
- A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).
- Triggers could be defined on the table, view, schema, or database with which the event is associated.



```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
  Declaration-statements
BEGIN
  Executable-statements
EXCEPTION
  Exception-handling-statements
END;
```



```
SQL> set serveroutput on
SQL> CREATE OR REPLACE TRIGGER display_age_changes
BEFORE DELETE OR INSERT OR UPDATE ON customer
FOR EACH ROW
WHEN (NEW.ID > 0)
DECLARE
   age_diff number;
BEGIN
   age_diff := :NEW.age  - :OLD.age;
   dbms_output.put_line('Old age: ' || :OLD.age);
   dbms_output.put_line('New age: ' || :NEW.age);
   dbms_output.put_line('age difference: ' || age_diff);
END;
/

Trigger created.
```



```
SQL> set serveroutput on
SQL> DECLARE
        total_rows number(2);
     BEGIN
        UPDATE  customer
        SET age = age + 5;
        IF sql%notfound THEN
           dbms_output.put_line('no customers updated');
        ELSIF sql%found THEN
           total_rows := sql%rowcount;
           dbms_output.put_line( total_rows || ' customers updated ');
        END IF;
     END;
     /
Old age: 20
New age: 25
age difference: 5
Old age: 23
New age: 28
age difference: 5
Old age: 22
New age: 27
age difference: 5
Old age: 24
New age: 29
age difference: 5
Old age: 25
New age: 30
age difference: 5
Old age: 25
New age: 30
age difference: 5
Old age: 25
New age: 30
age difference: 5
7 customers updated

PL/SQL procedure successfully completed.
```

26

RESULT:
Thus the PL/SQL Triggers program was created and executed successfully.

EX.NO:9 DATE:	VIEW AND INDEX

AIM:

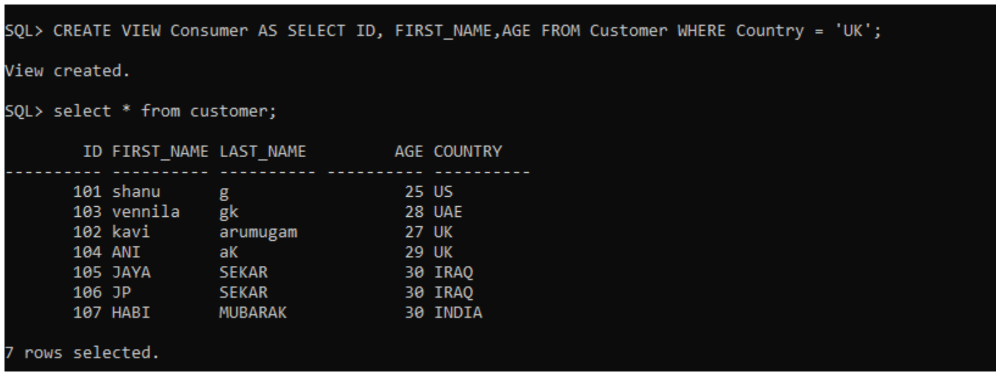
To create a view and index for database tables with a large number of records.

PROCEDURE:

CREATE VIEW Statement

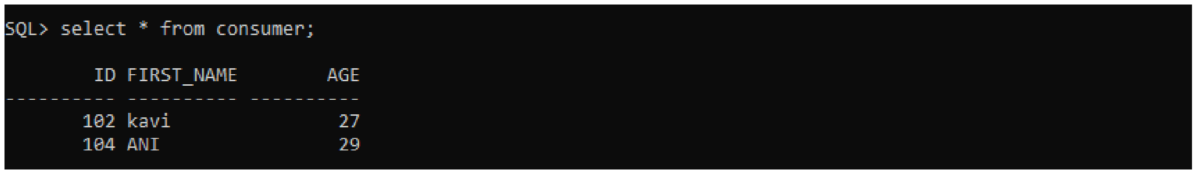
- In SQL, a view is a virtual table based on the result-set of an SQL statement.
- A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.
- You can add SQL statements and functions to a view and present the data as if the data were coming from one single table.

A view is created with the CREATE VIEW statement.



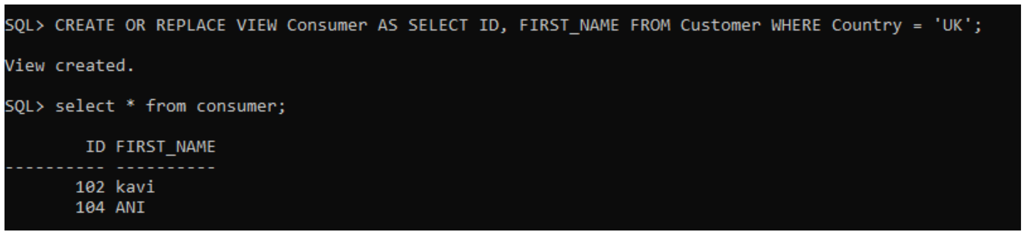
```
SQL> CREATE VIEW Consumer AS SELECT ID, FIRST_NAME,AGE FROM Customer WHERE Country = 'UK';

View created.

SQL> select * from customer;

        ID FIRST_NAME LAST_NAME         AGE COUNTRY
---------- ---------- ---------- ---------- ----------
       101 shanu      g                  25 US
       103 vennila    gk                 28 UAE
       102 kavi       arumugam           27 UK
       104 ANI        aK                 29 UK
       105 JAYA       SEKAR              30 IRAQ
       106 JP         SEKAR              30 IRAQ
       107 HABI       MUBARAK            30 INDIA

7 rows selected.
```

SELECT VIEW Statement



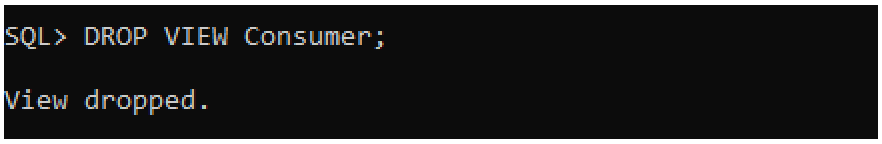
```
SQL> select * from consumer;

        ID FIRST_NAME        AGE
---------- ---------- ----------
       102 kavi               27
       104 ANI                29
```

REPLACE VIEW Statement



```
SQL> CREATE OR REPLACE VIEW Consumer AS SELECT ID, FIRST_NAME FROM Customer WHERE Country = 'UK';

View created.

SQL> select * from consumer;

        ID FIRST_NAME
---------- ----------
       102 kavi
       104 ANI
```

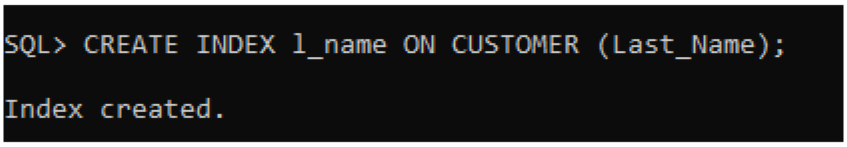
DROP VIEW Statement



```
SQL> DROP VIEW Consumer;

View dropped.
```

Index Create Statement

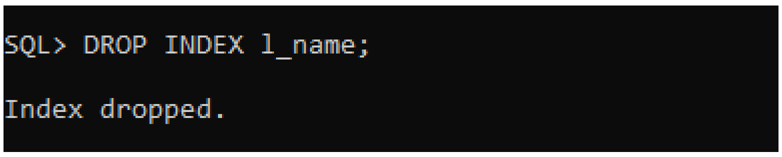
The SQL statement below creates an index named "l_name" on the "Last_Name" column in the "customer" table:



```
SQL> CREATE INDEX l_name ON CUSTOMER (Last_Name);

Index created.
```

Drop Index Statement

The DROP INDEX statement is used to delete an index in a table.



```
SQL> DROP INDEX l_name;

Index dropped.
```

RESULT:

Thus the View and Index SQL Program using executed successfully.

Ex No:10 **DATE:**	**XML DATABASE**

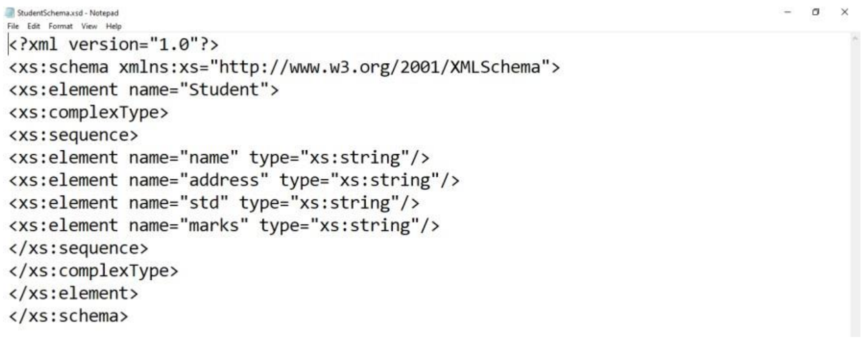
AIM:

To create an XML database to validate it using XML Schema.

PROCEDURE:

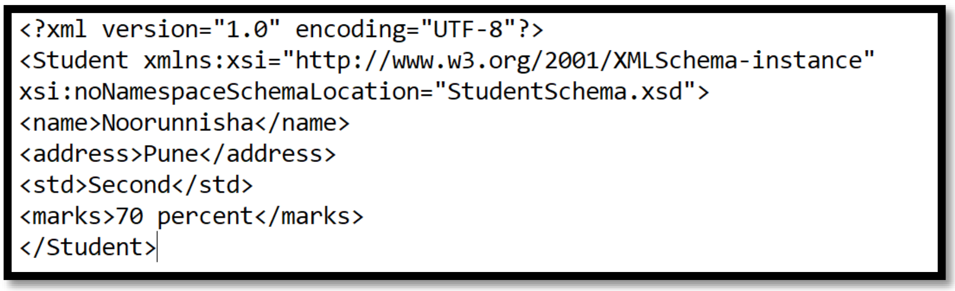
- The XML schemas are used to represent the **structure of XML document.**
- The goal or purpose of XML schema is to define the building blocks of an XML document. These can be used as analternative to XML DTD.
- The XML schema language is called as **XML Schema Definition (XSD) language.**
- XML schema defines elements, attributes, elements having child elements, order of child elements. It also definesfixed and default values of elements and attributes.
- XML schema also allows the developer to use **data types.**

XML Schema [StudentSchema.xsd]



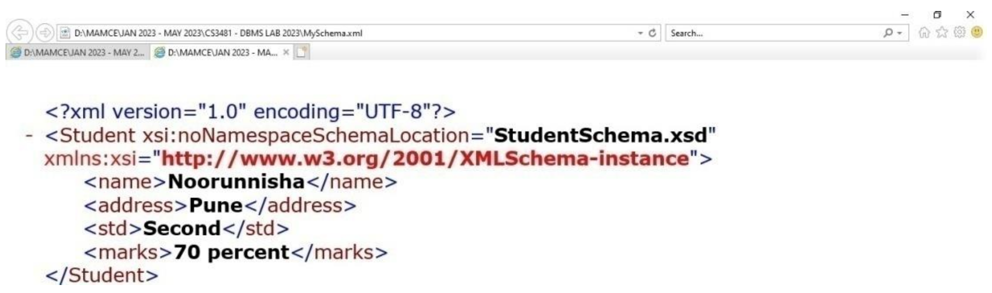
```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="Student">
<xs:complexType>
<xs:sequence>
<xs:element name="name" type="xs:string"/>
<xs:element name="address" type="xs:string"/>
<xs:element name="std" type="xs:string"/>
<xs:element name="marks" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

XML Document *[MySchema.xml]*



```
<?xml version="1.0" encoding="UTF-8"?>
<Student xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="StudentSchema.xsd">
<name>Noorunnisha</name>
<address>Pune</address>
<std>Second</std>
<marks>70 percent</marks>
</Student>
```

OUTPUT:



D:\MAMCE\JAN 2023 - MAY 2023\CS3481 - DBMS LAB 2023\MySchema.xml

```
  <?xml version="1.0" encoding="UTF-8"?>
- <Student xsi:noNamespaceSchemaLocation="StudentSchema.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <name>Noorunnisha</name>
      <address>Pune</address>
      <std>Second</std>
      <marks>70 percent</marks>
  </Student>
```

RESULT:

Thus the XML database was created and validated successfully.

EX.NO:11 **DATE:**	**NOSQL DATABASE TOOLS**

AIM:

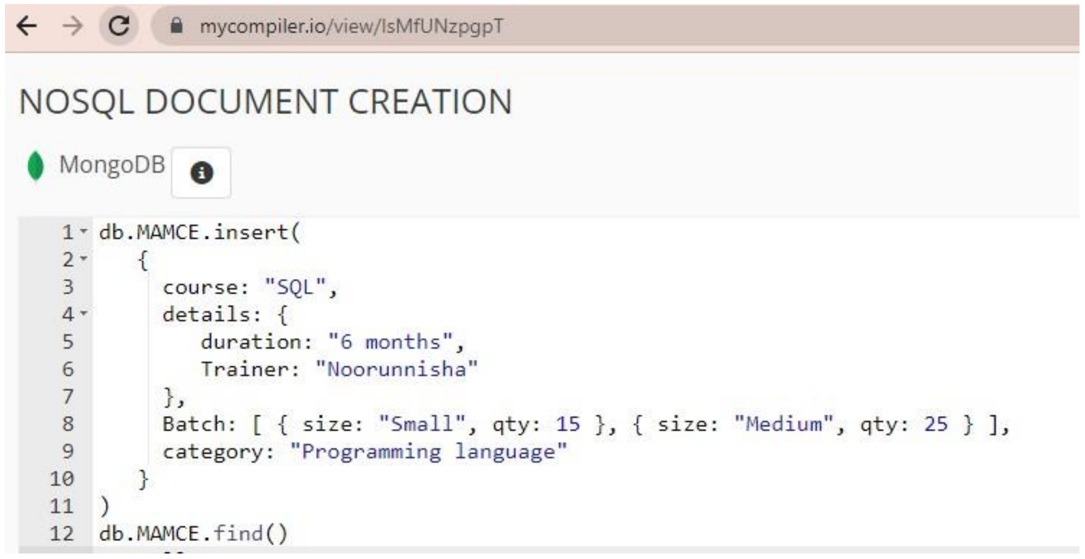
To create a Document, column and Graph based data using NOSQL Database tool.

PROCEDURE:

Document Oriented Database – NOSQL DATABASE TOOLS (MONGO DB)

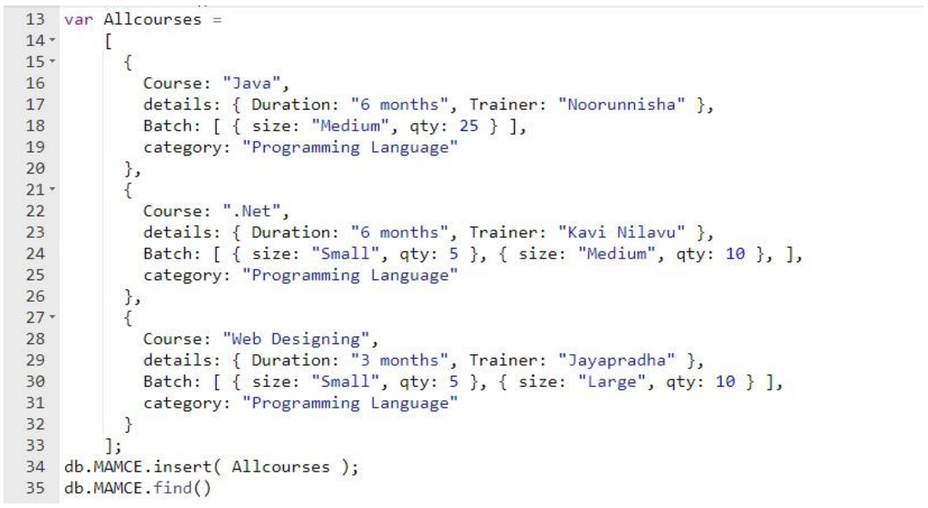
- A collection of documents
- Data in this model is stored inside documents.
- A document is a key value collection where the key allows access to its value.
- Documents are not typically forced to have a schema and therefore are flexible and easy to change.
- Documents are stored into collections in order to group different kinds of data.
- Documents can contain many different key-value pairs, or key-array pairs, or even nested documents.
- Here is a comparison between the classic relational model and the document model
- Example of Document Oriented databases : MongoDB, CouchDB etc.

11.1.2 Single Document Insertion Query

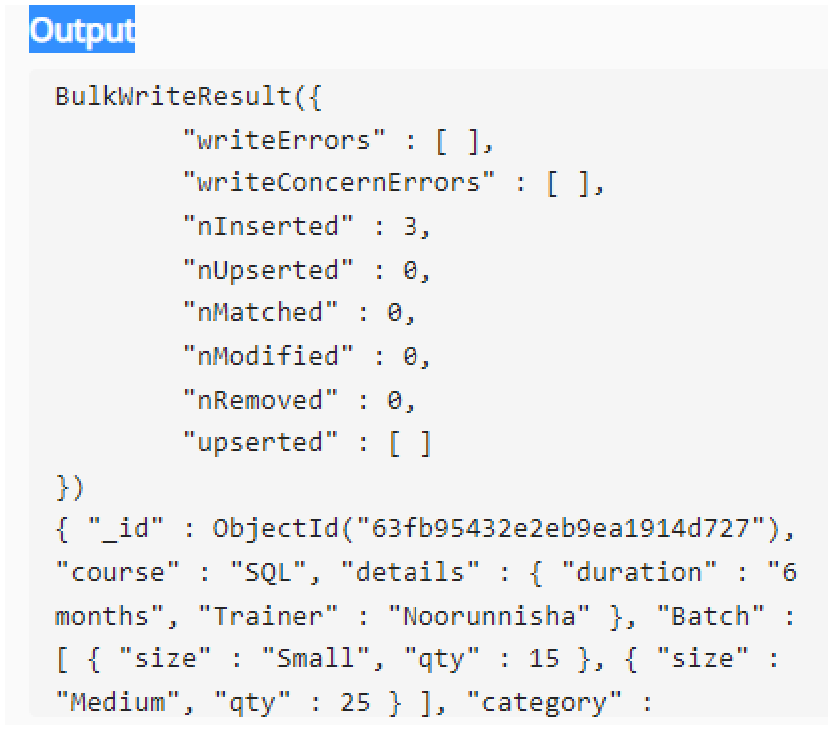


```
db.MAMCE.insert(
   {
     course: "SQL",
     details: {
        duration: "6 months",
        Trainer: "Noorunnisha"
     },
     Batch: [ { size: "Small", qty: 15 }, { size: "Medium", qty: 25 } ],
     category: "Programming language"
   }
)
db.MAMCE.find()
```

11.1.2 Multiple Document Insertion Query



```
var Allcourses =
    [
      {
        Course: "Java",
        details: { Duration: "6 months", Trainer: "Noorunnisha" },
        Batch: [ { size: "Medium", qty: 25 } ],
        category: "Programming Language"
      },
      {
        Course: ".Net",
        details: { Duration: "6 months", Trainer: "Kavi Nilavu" },
        Batch: [ { size: "Small", qty: 5 }, { size: "Medium", qty: 10 }, ],
        category: "Programming Language"
      },
      {
        Course: "Web Designing",
        details: { Duration: "3 months", Trainer: "Jayapradha" },
        Batch: [ { size: "Small", qty: 5 }, { size: "Large", qty: 10 } ],
        category: "Programming Language"
      }
    ];
db.MAMCE.insert( Allcourses );
db.MAMCE.find()
```



Output

```
BulkWriteResult({
        "writeErrors" : [ ],
        "writeConcernErrors" : [ ],
        "nInserted" : 3,
        "nUpserted" : 0,
        "nMatched" : 0,
        "nModified" : 0,
        "nRemoved" : 0,
        "upserted" : [ ]
})
{ "_id" : ObjectId("63fb95432e2eb9ea1914d727"),
"course" : "SQL", "details" : { "duration" : "6
months", "Trainer" : "Noorunnisha" }, "Batch" :
[ { "size" : "Small", "qty" : 15 }, { "size" :
"Medium", "qty" : 25 } ], "category" :
```

Column Based Database – NOSQL DATABASE TOOLS (CASSENDRA / SQLLite)

- Column-oriented databases primarily work on columns and every column is treated individually.
- Values of a single column are stored contiguously.
- Column stores data in column specific files.
- In Column stores, query processors work on columns too.
- All data within each column data file have the same type which makes it ideal for compression.
- Column stores can improve the performance of queries as it can access specific column data.
- Example of Column-oriented databases : BigTable, Cassandra, SimpleDB etc.

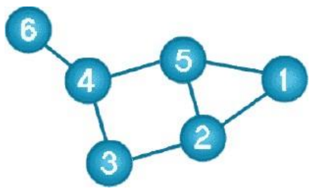
```
-- create
CREATE TABLE MAMCE_EMPLOYEE (
  empId int PRIMARY KEY,
  name text,
  dept text
);

-- insert
INSERT INTO MAMCE_EMPLOYEE(empId,name,dept) VALUES (0001, 'Nisha', 'Associate Professor');
INSERT INTO MAMCE_EMPLOYEE(empId,name,dept) VALUES (0002, 'Noor', 'Assistant Professor');
INSERT INTO MAMCE_EMPLOYEE(empId,name,dept) VALUES (0003, 'Habi', 'Lecturer');

-- fetch
SELECT * FROM MAMCE_EMPLOYEE;
```

OUTPUT:

```
1|Nisha|Associate Professor
2|Noor|Assistant Professor
3|Habi|Lecturer
```

Graph Based Database

A graph data structure consists of a finite (and possibly mutable) set of ordered pairs, called edges or arcs, of certain entities called nodes or vertices. The following picture presents a labelled graph of 6 vertices and 7 edges.



Example of Graph databases : OrientDB, Neo4J, Titan.etc.

OrientDB database is not only a Document database but also a Graph database. New concepts such as Vertex and Edge are used to store the data in the form of graph. It applies polymorphism on vertices. The base class for Vertex is V.

Vertex Creation

Execute the following query to create a vertex without ‘name’ and on the base class V.

```
orientdb> CREATE VERTEX
```

If the above query is executed successfully, you will get the following output.

```
Created vertex 'V#9:0 v1' in 0.118000 sec(s)
```

Execute the following query to create a new vertex class named v1, then create vertex in that class.

```
orientdb> CREATE CLASS V1 EXTENDS V
orientdb> CREATE VERTEX V1
```

```
Created vertex 'V1#14:0 v1' in 0.004000 sec(s)
```

Execute the following query to create a new vertex of the class named v1, defining its properties such as brand = 'Maruti'and name = 'Swift'.

```
orientdb> CREATE VERTEX V1 SET brand = 'maruti', name = 'swift'
```

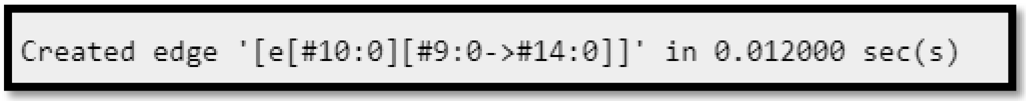
```
Created vertex 'V1#14:1{brand:maruti,name:swift} v1' in 0.004000 sec(s)
```

Edge Creation

Execute the following query to create an edge E between two vertices #9:0 and #14:0.

```
orientdb> CREATE EDGE FROM #11:4 TO #13:2
```

If the above query is executed successfully, you will get the following output.



```
Created edge '[e[#10:0][#9:0->#14:0]]' in 0.012000 sec(s)
```

RESULT:

Thus the NOSQL database tools are used to Document based, Column Based and Graph based databases are created andexecuted successfully.

EX.NO:12 **DATE:**	**GUI BASED DATABASE APPLICATION**

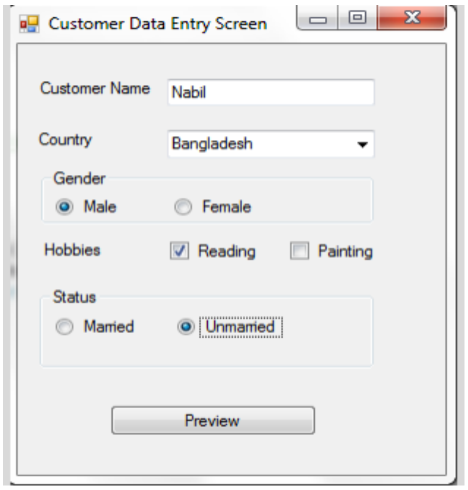
AIM:

To create a Simple GUI based database application and incorporate all the features.

PROCEDURE:

CREATING A SIMPLE CUSTOMER SCREEN WHICH TAKES CUSTOMER NAME, COUNTRY, GENDER, HOBBY ANDSTATUS

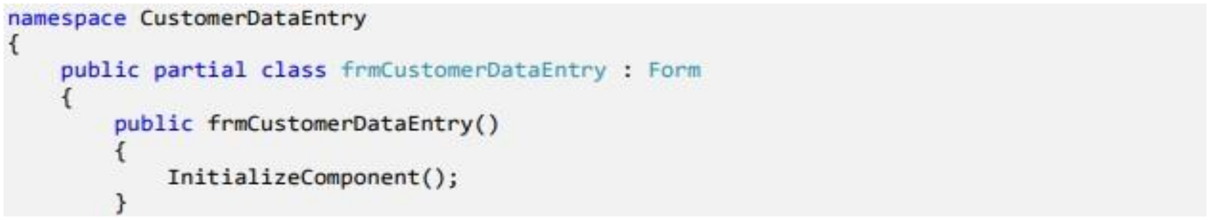
1. Create a Windows Form and in the 'Text' Properties of the Form write: Customer Data Entry Screen
2. Add Labels from the Tool Box and Add corresponding Text Boxes for Name and Country field.
3. Use radioButton when you want to select a single option from multiple choices.
4. To keep the radio Buttons in a group, first Drag a GroupBox and inside it drag the radioButtons.
5. Use checkbox when you want to select multiple options from a set of choices.
6. Give unique names for each radioButtons and checkboxes
7. In the following form radioButton for Male is named as radioMale and for Female as radioFemale
8. Similarly radioButton for Married is named as radioMarried and for Unmarried as radioUnmarried.
9. Finally, a Preview Button is added at the bottom of the form which when clicked will show the given data inanother form.
10. Name the preview button as btnPreview.the checkbox for Reading and Painting are named as chkReading andchkPainting



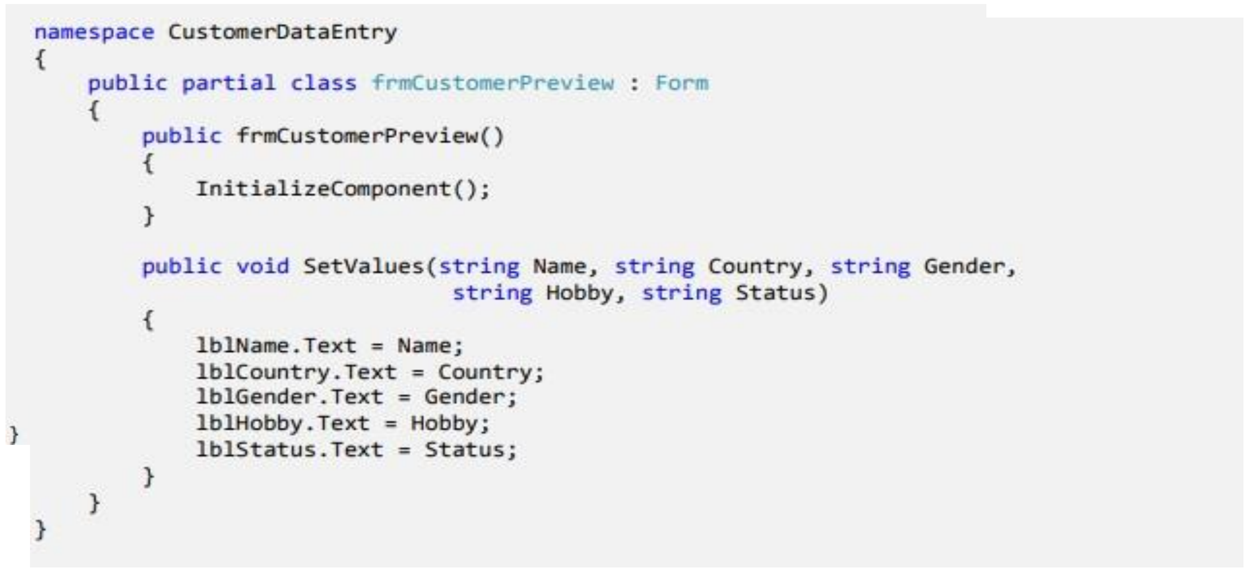
Customer Name: Nabil
Country: Bangladesh
Gender: ● Male ○ Female
Hobbies: ☑ Reading ☐ Painting
Status: ○ Married ● Unmarried

CREATING A PREVIEW SCREEN THAT WILL DISPLAY DATA ENTERED IN TO THE CUSTOMER DATA ENTRY SCREEN

1. In the Customer Data Entry form, double click the button Preview and write down the functionality of the clicking event that is show a form which will contain 5 labels for the titles and another 5 Labels to show the data that was given as input.
2. Write a user-defined function SetValues (. . .) that sets the value of the given input to the Labels.

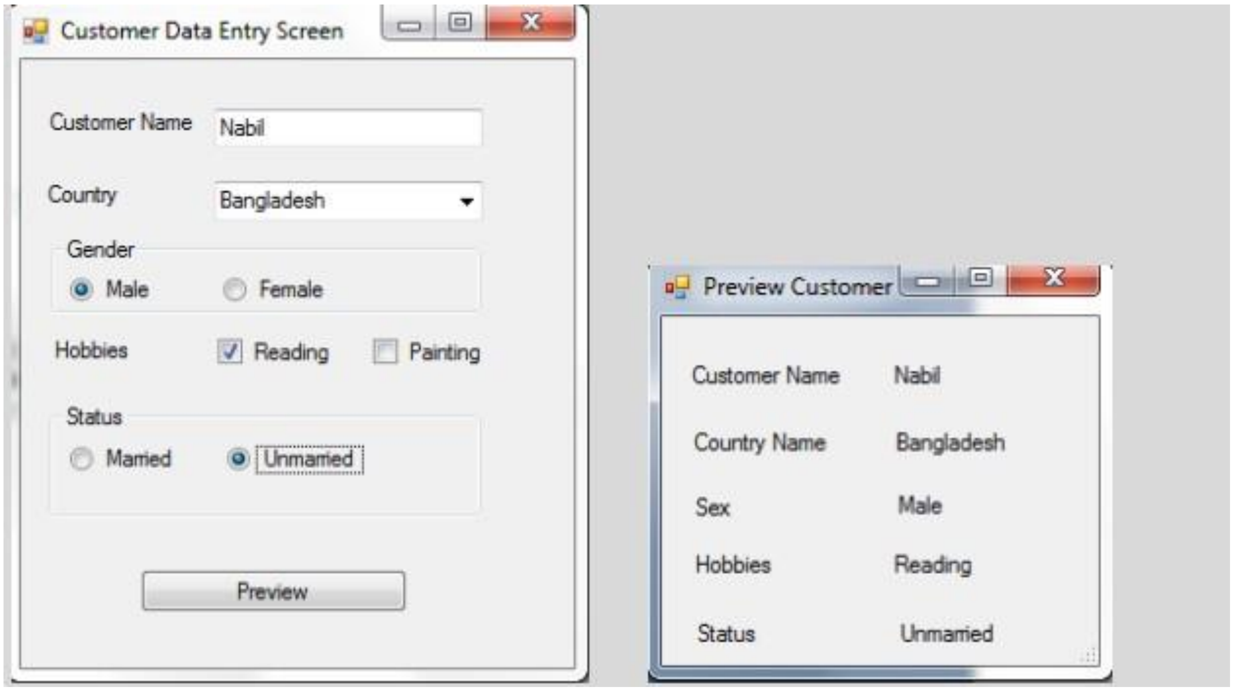


```
namespace CustomerDataEntry
{
    public partial class frmCustomerDataEntry : Form
    {
        public frmCustomerDataEntry()
        {
            InitializeComponent();
        }
```



```
namespace CustomerDataEntry
{
    public partial class frmCustomerPreview : Form
    {
        public frmCustomerPreview()
        {
            InitializeComponent();
        }

        public void SetValues(string Name, string Country, string Gender,
                              string Hobby, string Status)
        {
            lblName.Text = Name;
            lblCountry.Text = Country;
            lblGender.Text = Gender;
            lblHobby.Text = Hobby;
            lblStatus.Text = Status;
        }
    }
}
```

```
}
```



Customer Name: Nabil

Country: Bangladesh

Gender: [● Male] ○ Female

Hobbies: [✓ Reading] ☐ Painting

Status: ○ Married [● Unmarried]

Customer Name	Nabil
Country Name	Bangladesh
Sex	Male
Hobbies	Reading
Status	Unmarried

CREATING NAVIGATIONAL MENUS WHICH WILL HELP US TO NAVIGATE CUSTOMER ENTRY SCREEN AND DISPLAYSCREENS EASILY

1. To create a user friendly interface we can use something called MDIParent Form and name it as MDICustomer.

2

. View Solution Explorer Select your project Right Click Add New Item Choose MDIParent Form and then a
it.

3. In the Design View of MDIParent, Click File and you will find some menu options

4. Edit a menu name and give it Enter Customer 5. Then double click the Enter Customer menu

6. When this menu is clicked you have to invoke the Customer Data Entry Form which is named here asfrmCustomerDataEntry

7. To keep the form within the MDIParent select the obj.MdiParent = this;

8. To show the form use the function Show ()

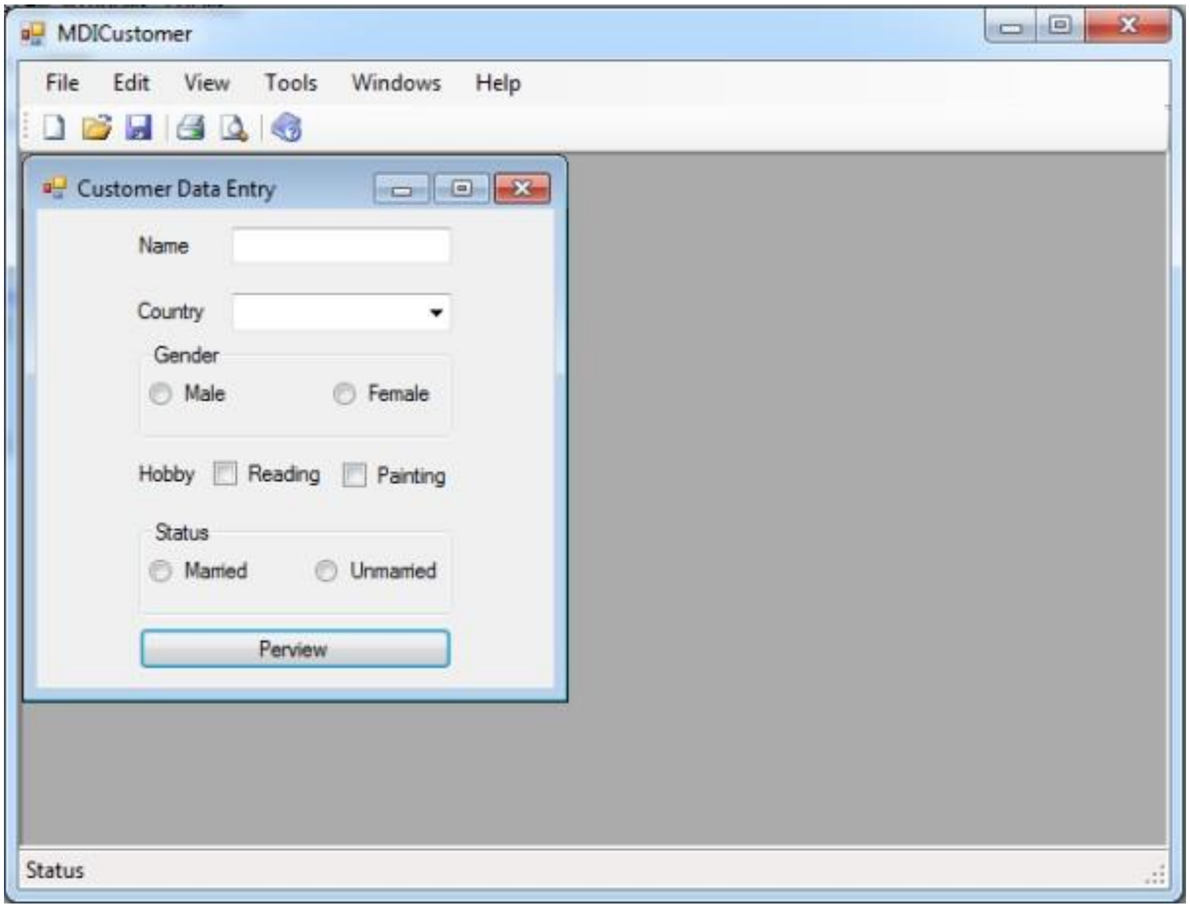
9. In the program.cs file specify the form MDICustomer. This will invoke the MDICustomer form when the application isrun.

In MDI Parent form:

```
private void customerDataEntryToolStripMenuItem_Click(object sender, EventArgs e)
{
    frmCustomerDataEntry obj = new frmCustomerDataEntry();
    obj.MdiParent = this;
    obj.Show();
}
```

In program.cs:

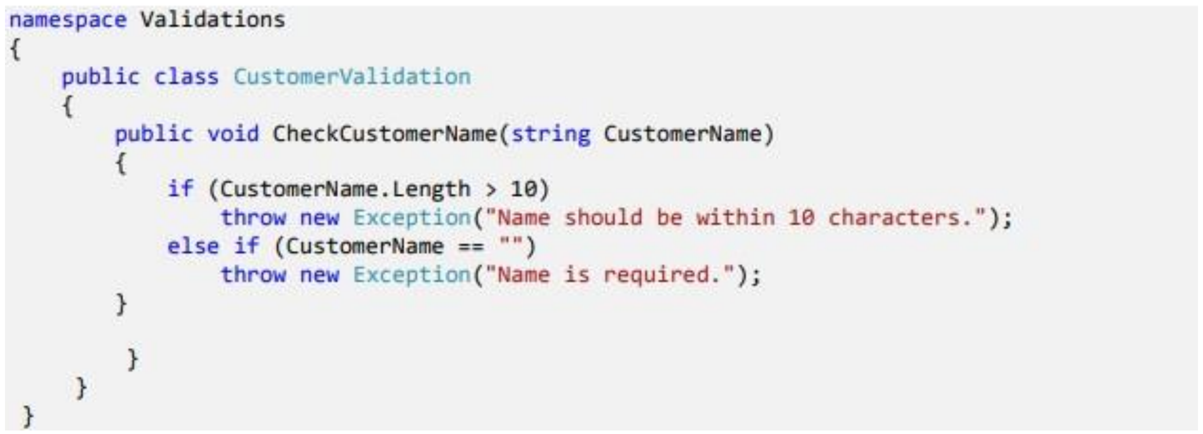
```
static void Main()
{
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);
    Application.Run(new MDICustomer());
}
```



REUSING CODE BY CREATING CLASSES AND OBJECTS AND FORM VALIDATION

1. An important feature of programming is to avoid the reusability of code.
2. To write the validations of different types, create a new class.
3. View ◊ Solution Explorer ◊ Select your project ◊ Right Click ◊ Add ◊ New Item ◊ Add a new Class and name it asCustomerValidation.cs
4. In CustomerValidation.cs you can define various functions with different names and parameters. An example is givenbelow –

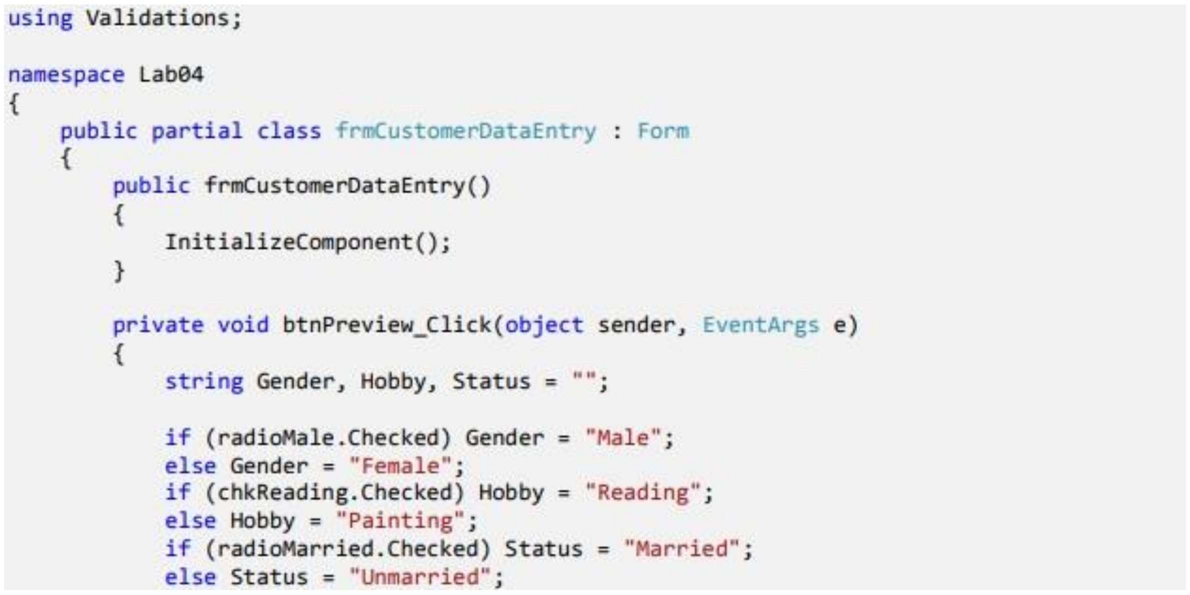
In project Validations:



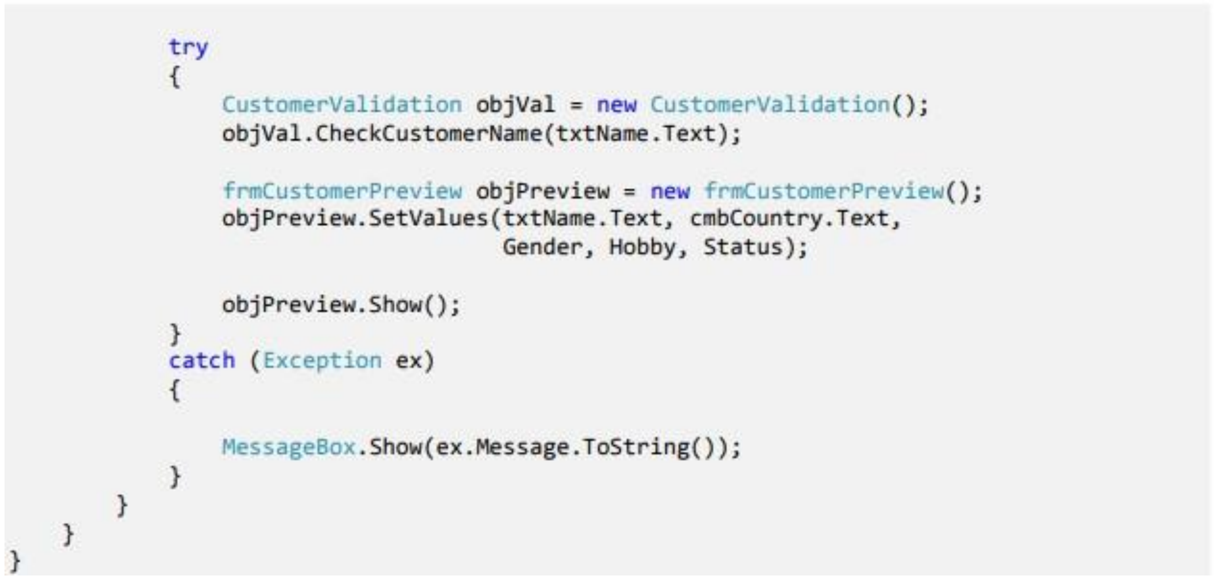
```
namespace Validations
{
    public class CustomerValidation
    {
        public void CheckCustomerName(string CustomerName)
        {
            if (CustomerName.Length > 10)
                throw new Exception("Name should be within 10 characters.");
            else if (CustomerName == "")
                throw new Exception("Name is required.");
        }

         }
     }
 }
```

In frmCustomerDataEntry:

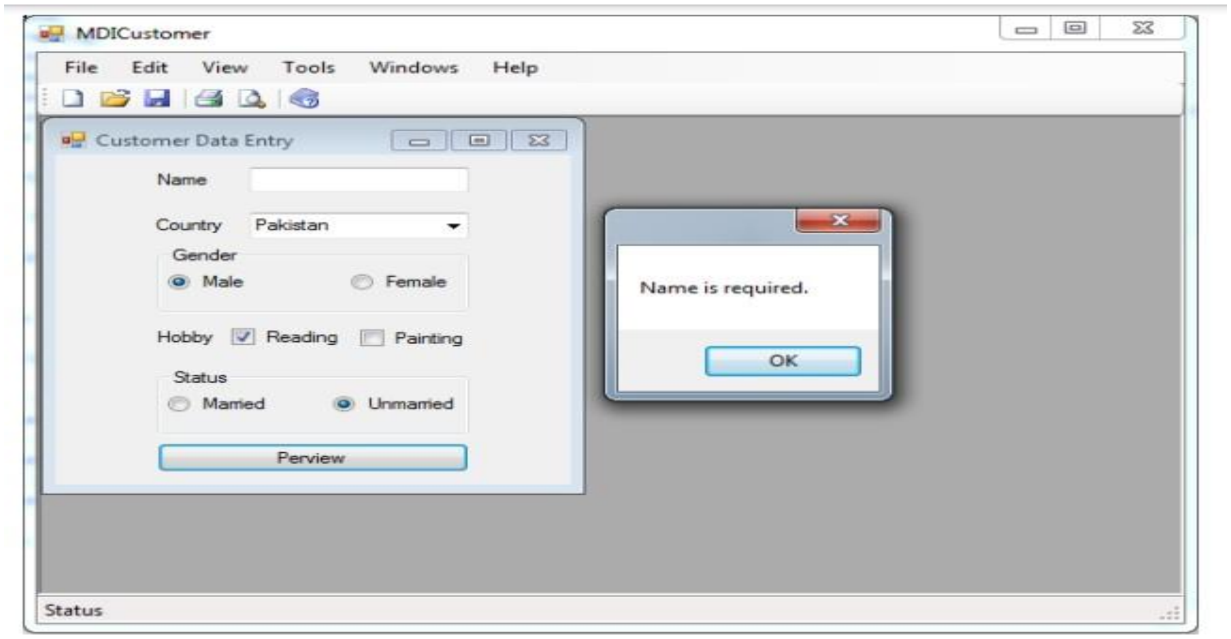
1. In the Preview button Click event we have write some code to specify that if the user does not enter any CustomerName then an error message should be displayed.
2. We have to call the CheckCustomerName (string) function in the CustomerValidation class.
3. This code is specified in the try-catch block below -



```
using Validations;

namespace Lab04
{
    public partial class frmCustomerDataEntry : Form
    {
        public frmCustomerDataEntry()
        {
            InitializeComponent();
        }

        private void btnPreview_Click(object sender, EventArgs e)
        {
            string Gender, Hobby, Status = "";

            if (radioMale.Checked) Gender = "Male";
            else Gender = "Female";
            if (chkReading.Checked) Hobby = "Reading";
            else Hobby = "Painting";
            if (radioMarried.Checked) Status = "Married";
            else Status = "Unmarried";
```



```
            try
            {
                CustomerValidation objVal = new CustomerValidation();
                objVal.CheckCustomerName(txtName.Text);

                frmCustomerPreview objPreview = new frmCustomerPreview();
                objPreview.SetValues(txtName.Text, cmbCountry.Text,
                                     Gender, Hobby, Status);

                objPreview.Show();
            }
            catch (Exception ex)
            {

                MessageBox.Show(ex.Message.ToString());
            }
        }
    }
}
```



Name is required.

CONNECTING TO SQL SERVER, GETTING DATA AND GETTING ACQUAINTED WITH ADO.NET COMPONENTS, LIKE CONNECTION, COMMAND AND DATA READER

As C# is a Microsoft Product, it is better to use a Microsoft Data management software which is SQL Server.

- ADO.NET (ActiveX Data Object) helps to connect the User Interface to the Database and also sends and retrieves datafrom SQL Server
- To Create a Database connection you have to follow the steps below –

1. View Server Explorer Select Data Connections Right Click Create New SQL Server Database Server Name: “.\sqlexpress ” Give a Database Name: “CustomerDB” Ok

2. View ToolBox Data Data GridView Drag it to the frmCustomerDataEntry form and name it as dtgCustomer.

3. View Server Explorer Here your connection is shown under Data Connection. Select that new connection Right Click Properties Connection String (This string is needed in the code below)

4. To create a new Table in the Database you have to select your new connection and under it there is an option Tables.Right Click on the Tables Create New Table

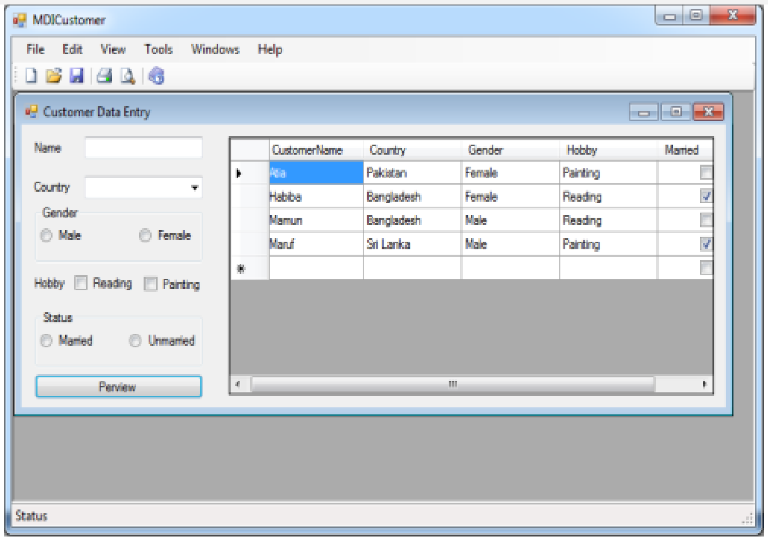
5. Save this and give a name (here CustomerDB is given) to the Table

6. Now that your Table has been created, you can add different columns: Select your Table Right Click Show Table Definition Now add column names with respecti ve datatypes

7. You can add data in your Table: Select your Table Right Click Show Table Data

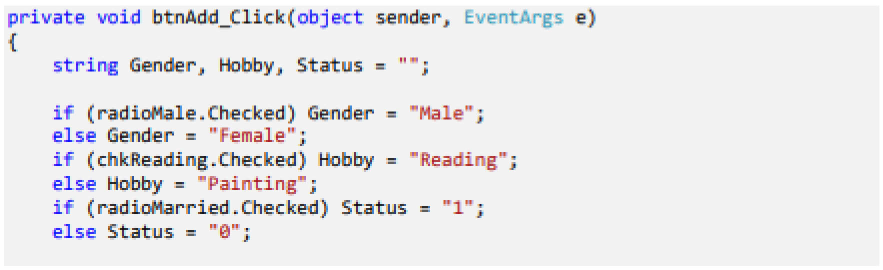
```
private void frmCustomerDataEntry_Load(object sender, EventArgs e)
{
    loadCustomer();
}

private void loadCustomer()
{
    // Open a Connection
    string strConnection = "Data Source=.\\sqlexpress;Initial Catalog=CustomerDB;"
                         + "Integrated Security=True";
    SqlConnection objConnection = new SqlConnection(strConnection);
    objConnection.Open();

    // Fire a Command
    string strCommand = "Select * From CustomerTable";
    SqlCommand objCommand = new SqlCommand(strCommand, objConnection);
```

```
    // Bind Data with UI
    DataSet objDataSet = new DataSet();
    SqlDataAdapter objAdapter = new SqlDataAdapter(objCommand);
    objAdapter.Fill(objDataSet);
    dtgCustomer.DataSource = objDataSet.Tables[0];

    // Close the Connection
    objConnection.Close();
}
```

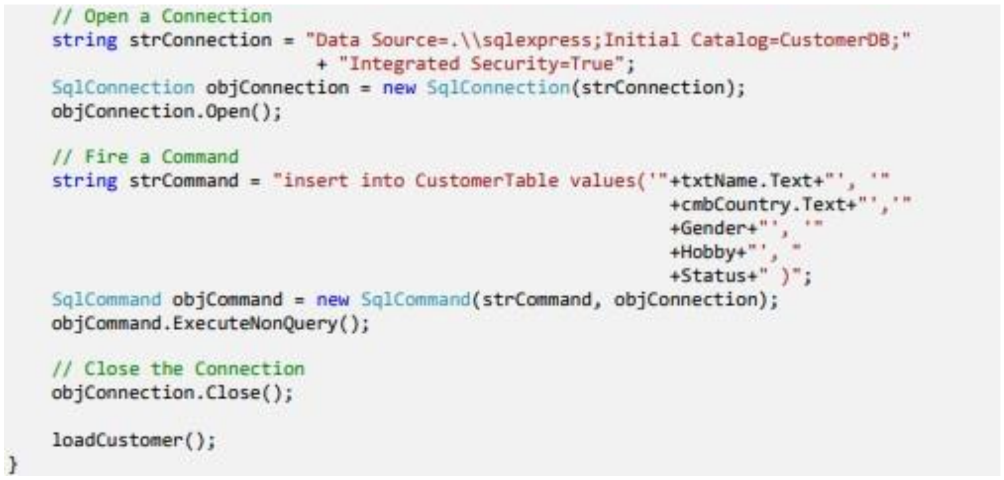


CustomerName	Country	Gender	Hobby	Married
Alia	Pakistan	Female	Painting	☐
Habiba	Bangladesh	Female	Reading	✓
Mamun	Bangladesh	Male	Reading	☐
Maruf	Sri Lanka	Male	Painting	✓
				☐

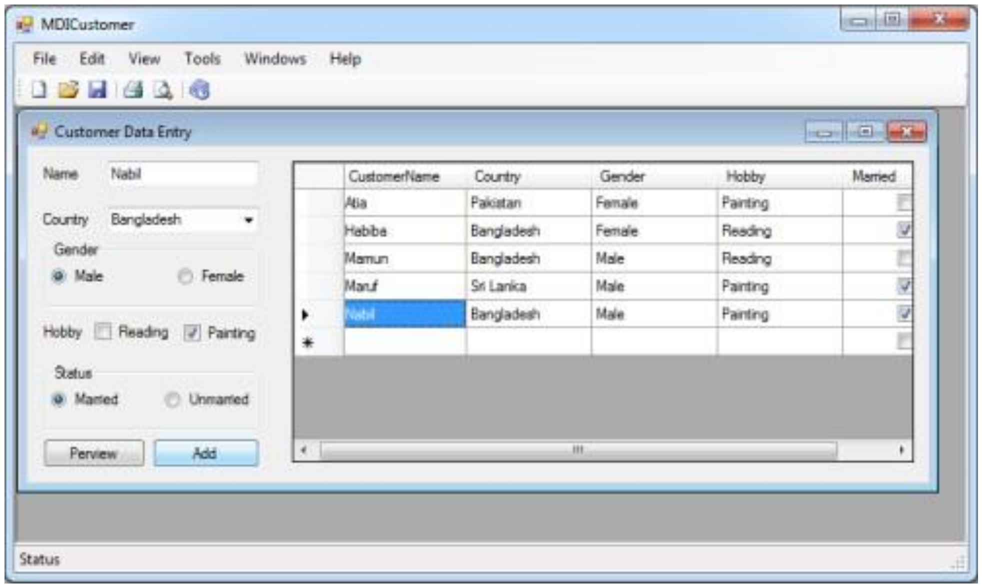
: INSERTING INTO DATABASE AND DYNAMIC CONNECTION STRING



```
private void btnAdd_Click(object sender, EventArgs e)
{
    string Gender, Hobby, Status = "";

    if (radioMale.Checked) Gender = "Male";
    else Gender = "Female";
    if (chkReading.Checked) Hobby = "Reading";
    else Hobby = "Painting";
    if (radioMarried.Checked) Status = "1";
    else Status = "0";
```



```
    // Open a Connection
    string strConnection = "Data Source=.\\sqlexpress;Initial Catalog=CustomerDB;"
                         + "Integrated Security=True";
    SqlConnection objConnection = new SqlConnection(strConnection);
    objConnection.Open();

    // Fire a Command
    string strCommand = "insert into CustomerTable values('"+txtName.Text+"', '"
                                                          +cmbCountry.Text+"','"
                                                          +Gender+"', '"
                                                          +Hobby+"', "
                                                          +Status+" )";
    SqlCommand objCommand = new SqlCommand(strCommand, objConnection);
    objCommand.ExecuteNonQuery();

    // Close the Connection
    objConnection.Close();

    loadCustomer();
}
```



Name: Nabil

Country: Bangladesh

Gender: [✓ Male] Female

Hobby: Reading [✓ Painting]

Status: [✓ Married] Unmarried

	CustomerName	Country	Gender	Hobby	Married
	Atia	Pakistan	Female	Painting	☐
	Habiba	Bangladesh	Female	Reading	✓
	Mamun	Bangladesh	Male	Reading	☐
	Maruf	Sri Lanka	Male	Painting	✓
▸	Nabil	Bangladesh	Male	Painting	✓
*					☐

RESULT:

Thus the Visual Studio C# Program used to Database Application GUI was designed successfully.

EX.NO:13 **DATE:**	**CASE STUDY**

AIM:
To design database using ER modeling, normalization constraints and to implement the Operations of Bank Management System using the visual basic as front end and oracle as back end todesign a forms.

PROCEDURE:
<u>Database design using E-R model and Normalization</u>

SALESMAN (*Salesman_id, Name, City, Commission*) CUSTOMER (*Customer_id, Cust_Name*, *City*, *Grade*, *Salesman_id*)

ORDERS (*Ord_No*, *Purchase_Amt*, *Ord_Date*, *Customer_id*, *Salesman_id*)

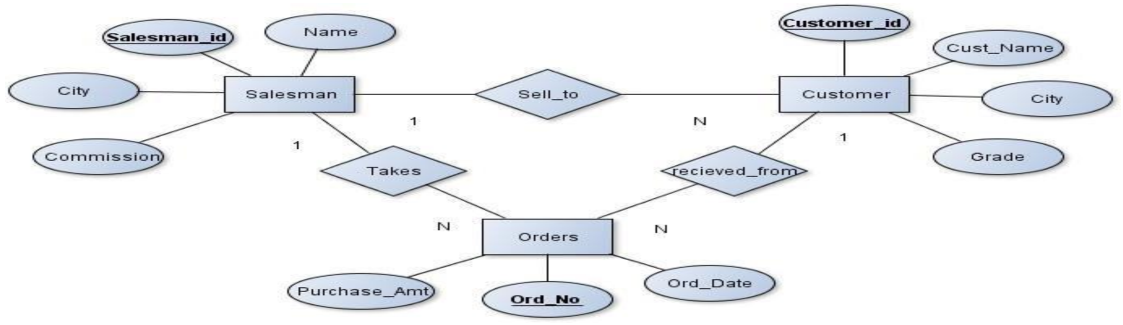
Write SQL queries to

1. Count the customers with grades above Bangalore's average.
2. Find the name and numbers of all salesmen who had more than one customer.
3. List all salesmen and indicate those who have and don't have customers in their cities (Use UNION operation.)
4. Create a view that finds the salesman who has the customer with the highest order of a day.
5. Demonstrate the DELETE operation by removing salesman with id 1000. All his orders must also be deleted.

Representing Relationships

•**1:1** Relationships. The key of one relation is stored in the second relation. Look at examplequeries to determine which keyis queried most often.
•**1:N** Relationships. **Parent** - Relation on the "1" side. **Child** - Relation on the "Many" side.
•Represent each Entity as a relation. Copy the key of the parent into the child relation.
•**M:N** Solution: Introduce a third *Intersection relation* and copy keys from original two relations.
•Relationships. Many to Many relationships can not be directly implemented in relations.
•

ER MODEL DIAGRAM



CREATE TABLE SALESMAN (SALESMAN_ID NUMBER (4), NAME VARCHAR2 (20), CITY VARCHAR2 (20), COMMISSION VARCHAR2 (20), PRIMARY KEY (SALESMAN_ID));

CREATE TABLE CUSTOMER1 (CUSTOMER_ID NUMBER (4), CUST_NAME VARCHAR2 (20), CITY VARCHAR2 (20), GRADE NUMBER (3), PRIMARY KEY (CUSTOMER_ID), SALESMAN_ID REFERENCES SALESMAN (SALESMAN_ID) ON DELETE SET NULL);

CREATE TABLE ORDERS (ORD_NO NUMBER (5), PURCHASE_AMT NUMBER (10, 2), ORD_DATE DATE, PRIMARY KEY (ORD_NO), CUSTOMER_ID REFERENCES CUSTOMER1 (CUSTOMER_ID) ON DELETE CASCADE, SALESMAN_ID REFERENCES SALESMAN (SALESMAN_ID) ON DELETE CASCADE);

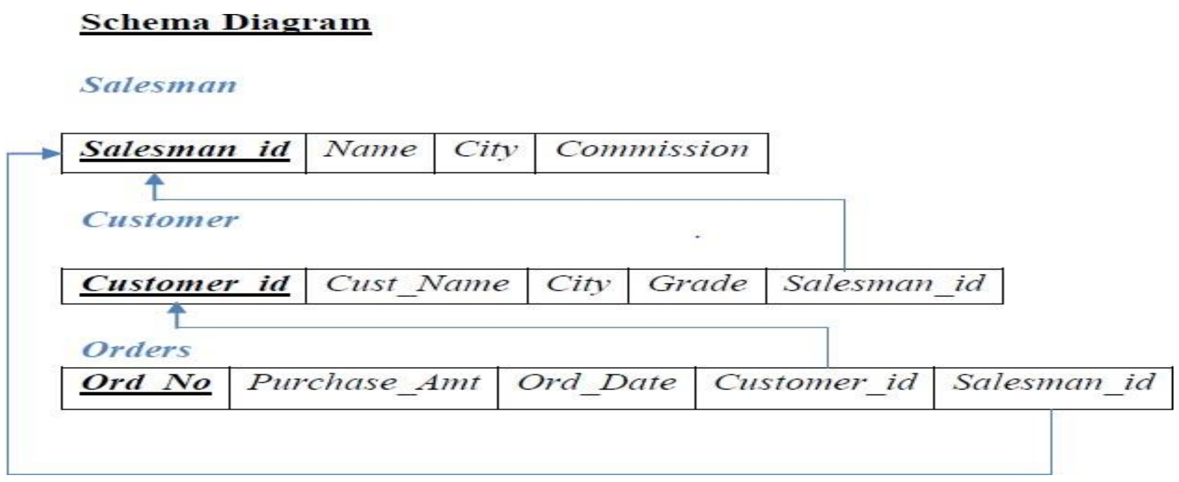
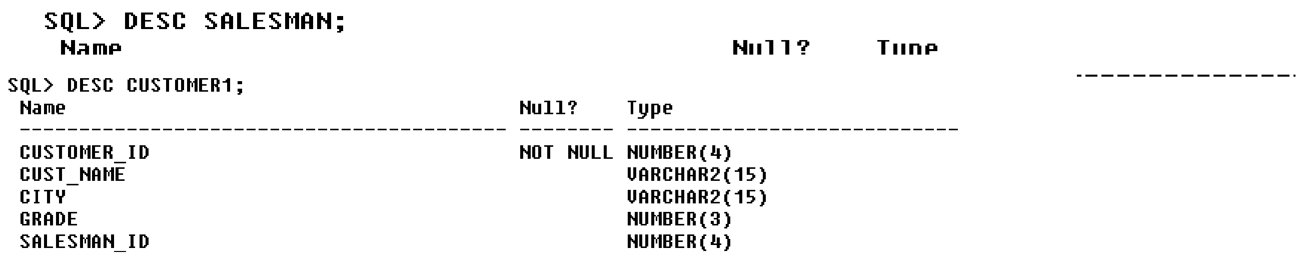
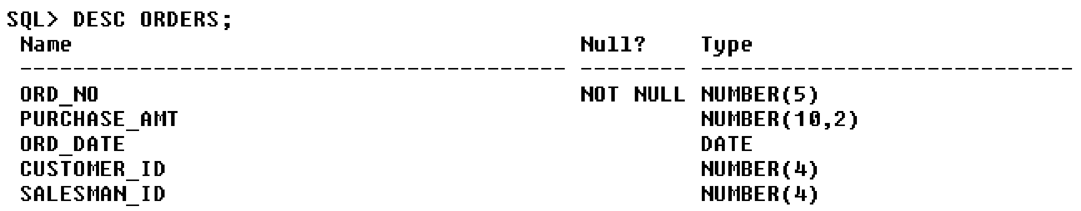


Table DescriptionsDESC SALESMAN;


```
SQL> DESC SALESMAN;
 Name                                          Null?     Tune
SQL> DESC CUSTOMER1;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 CUSTOMER_ID                               NOT NULL NUMBER(4)
 CUST_NAME                                          VARCHAR2(15)
 CITY                                               VARCHAR2(15)
 GRADE                                              NUMBER(3)
 SALESMAN_ID                                        NUMBER(4)
```

DESC CUSTOMER1;DESC ORDERS;



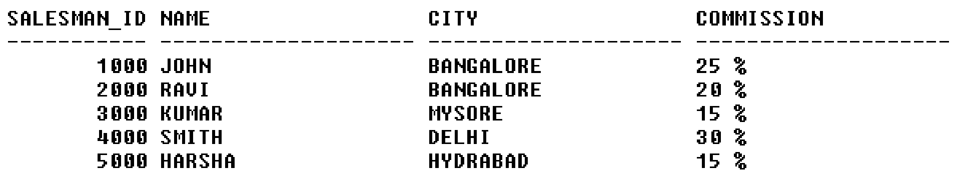
```
SQL> DESC ORDERS;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ORD_NO                                    NOT NULL NUMBER(5)
 PURCHASE_AMT                                       NUMBER(10,2)
 ORD_DATE                                           DATE
 CUSTOMER_ID                                        NUMBER(4)
 SALESMAN_ID                                        NUMBER(4)
```

Insertion of Values to Tables

INSERT INTO SALESMAN VALUES (1000, _JOHN‘,‘BANGALORE‘,‘25 %‘);
INSERT INTO CUSTOMER1 VALUES (10,

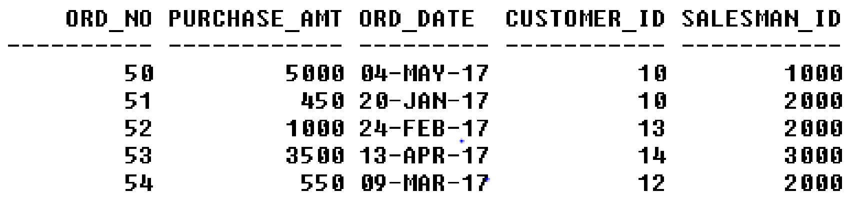
_PREETHI‘,‘BANGALORE‘, 100, 1000);

INSERT INTO ORDERS VALUES (50, 5000, _04-MAY-17‘, 10, 1000);SELECT * FROM SALESMAN;



```
SALESMAN_ID NAME                 CITY                 COMMISSION
----------- -------------------- -------------------- --------------------
       1000 JOHN                 BANGALORE            25 %
       2000 RAVI                 BANGALORE            20 %
       3000 KUMAR                MYSORE               15 %
       4000 SMITH                DELHI                30 %
       5000 HARSHA               HYDRABAD             15 %
```

SELECT * FROM ORDERS;

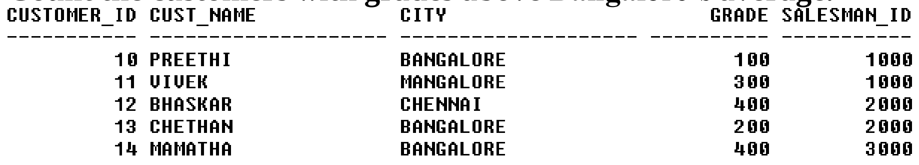


```
    ORD_NO PURCHASE_AMT ORD_DATE  CUSTOMER_ID SALESMAN_ID
---------- ------------ --------- ----------- -----------
        50         5000 04-MAY-17          10        1000
        51          450 20-JAN-17          10        2000
        52         1000 24-FEB-17          13        2000
        53         3500 13-APR-17          14        3000
        54          550 09-MAR-17          12        2000
```

Queries:

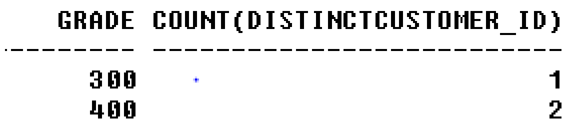
SELECT * FROM CUSTOMER1;

Count the customers with grades above Bangalore's average.



```
CUSTOMER_ID CUST_NAME            CITY                      GRADE SALESMAN_ID
----------- -------------------- -------------------- ---------- -----------
         10 PREETHI              BANGALORE                   100        1000
         11 VIVEK                MANGALORE                   300        1000
         12 BHASKAR              CHENNAI                     400        2000
         13 CHETHAN              BANGALORE                   200        2000
         14 MAMATHA              BANGALORE                   400        3000
```

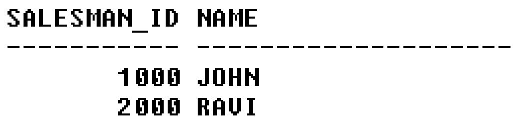
SELECT GRADE, COUNT (DISTINCT CUSTOMER_ID) FROM CUSTOMER1 GROUP BY GRADE HAVING GRADE > (SELECT AVG(GRADE) FROM CUSTOMER1 WHERE CITY='BANGALORE');



```
GRADE COUNT(DISTINCTCUSTOMER_ID)
-------- --------------------------
  300                           1
  400                           2
```

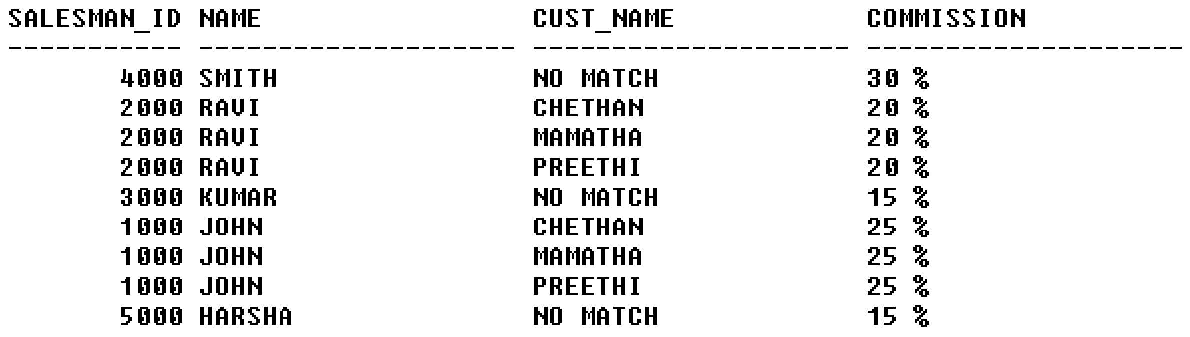
1. Find the name and numbers of all salesmen who had more than one customer.

SELECT SALESMAN_ID, NAME FROM SALESMAN A WHERE 1 < (SELECT COUNT (*) FROM CUSTOMER1 WHERESALESMAN_ID=A.SALESMAN_ID);



```
SALESMAN_ID NAME
----------- --------------------
       1000 JOHN
       2000 RAVI
```

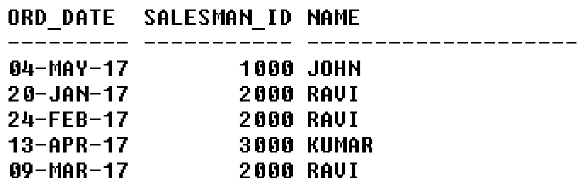
2. List all salesmen and indicate those who have and don’t have customers in their cities (UseUNION operation.) SELECT SALESMAN.SALESMAN_ID, NAME, CUST_NAME, COMMISSION FROM SALESMAN, CUSTOMER1 WHERE SALESMAN.CITY = CUSTOMER1.CITY UNION SELECT SALESMAN_ID, NAME, 'NO MATCH', COMMISSION FROM SALESMAN WHERE NOT CITY = ANY (SELECT CITY FROM CUSTOMER1) ORDER BY 2 DESC;



```
SALESMAN_ID NAME                 CUST_NAME            COMMISSION
----------- -------------------- -------------------- --------------------
       4000 SMITH                NO MATCH             30 %
       2000 RAVI                 CHETHAN              20 %
       2000 RAVI                 MAMATHA              20 %
       2000 RAVI                 PREETHI              20 %
       3000 KUMAR                NO MATCH             15 %
       1000 JOHN                 CHETHAN              25 %
       1000 JOHN                 MAMATHA              25 %
       1000 JOHN                 PREETHI              25 %
       5000 HARSHA               NO MATCH             15 %
```

3. Create a view that finds the salesman who has the customer with the highest order of a day.

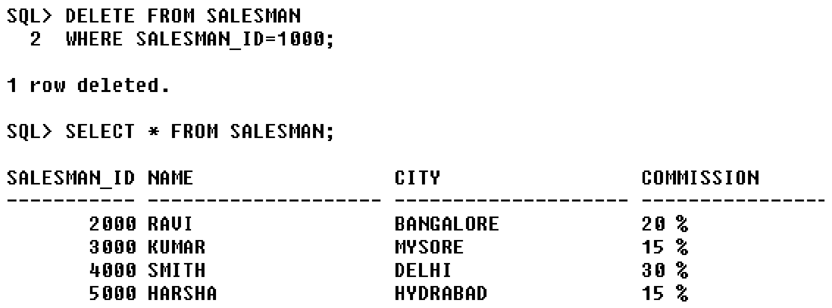
CREATE VIEW ELITSALESMAN AS SELECT B.ORD_DATE, A.SALESMAN_ID, A.NAME FROM SALESMAN A, ORDERS B WHERE A.SALESMAN_ID = B.SALESMAN_IDAND B.PURCHASE_AMT=(SELECT MAX (PURCHASE_AMT) FROM ORDERS C
WHERE C.ORD_DATE = B.ORD_DATE);



```
ORD_DATE  SALESMAN_ID NAME
--------- ----------- --------------------
04-MAY-17        1000 JOHN
20-JAN-17        2000 RAVI
24-FEB-17        2000 RAVI
13-APR-17        3000 KUMAR
09-MAR-17        2000 RAVI
```

4. Demonstrate the DELETE operation by removing salesman with id 1000. All his orders must also be deleted.

Use ON DELETE CASCADE at the end of foreign key definitions while creating child table orders and then execute the following: DELETE FROM SALESMAN WHERE SALESMAN_ID=1000;



```
SQL> DELETE FROM SALESMAN
  2  WHERE SALESMAN_ID=1000;

1 row deleted.

SQL> SELECT * FROM SALESMAN;

SALESMAN_ID NAME                 CITY                 COMMISSION
----------- -------------------- -------------------- ----------------
       2000 RAVI                 BANGALORE            20 %
       3000 KUMAR                MYSORE               15 %
       4000 SMITH                DELHI                30 %
       5000 HARSHA               HYDRABAD             15 %
```

RESULT:

Thus the SQL Program used to databases are designed using ER modeling, normalization constraints and the Operations of Bank Management System using visual basic as front end and oracle as back end to design forms successfully.