Experiment No.: 04

Title: To use DML operations and SQL queries to Populate the database

Batch:SY-IT(B3) **Roll No.:16010423076** **Experiment No: 04**

Aim: To use DML operations and SQL queries to populate the database .

Resources needed: PostgreSQL PgAdmin4

Theory:

The Data Manipulation Language (DML) is used to populate the table with values, modify the table values and remove the rows of the table.

The DML statements
are: SELECT

INSERT
UPDATE
DELETE

Procedure:

```
CREATE TABLE products (
product_no integer,

name text,
price
numeric );
```

Let us consider the above products table

Inserting rows:
The INSERT command requires the table name and column values

```
INSERT INTO products VALUES (1, ’Cheese’, 9.99);
```

If we don’t have values for all the columns, you can omit some of them. In that case, the columns will be filled with their default values. For example:

```
INSERT INTO products (product_no, name) VALUES (1, ’Cheese’)
```

Updating the values:
The UPDATE command requires three pieces of information:

1. The name of the table and column to update
2. The new value of the column
3. Which row(s) to update

```
UPDATE products SET price = 10 WHERE price = 5;
UPDATE products SET price = price * 1.10;
```

Deleting rows:

The syntax of the DELETE command is similar to the UPDATE command.

```
DELETE FROM products WHERE price = 10;
```

Retrieving values:

The general syntax of the SELECT command
is SELECT select_list FROM table_expression

```
SELECT * FROM table1;
SELECT * FROM products WHERE price=10;
SELECT product_no, name FROM products WHERE price=10;
```

Example:

```
insert into department values('IT', 101, 'mumbai');
insert into department values('COMP', 102, 'mumbai');
insert into department values('ETRX', 103, 'delhi');
insert into department values('EXTC', 104, 'chennai');
insert into department values('account', 105, 'mumbai');

insert into employee values('anita','m','sharma','emp0001',20000,'mumbai',101);
insert into employee values('nita','g','patil','emp0004',10000,'mumbai',101);
insert into employee values('krupita','v','jetali','emp0003',20000,'delhi',103);
insert into employee values('juhi','r','verma','emp0002',15000,'delhi',104);
insert into employee  values('anita','m','sharma', 'emp0005',20000,'mumbai',104);

insert into project values( 1, 'mumbai','website',101);
insert into project values( 2, 'chennai','coding',101);
insert into project values( 3, 'mumbai','testing',102);
insert into project values( 4, 'delhi','documentaion',103);

insert into works_on values(1,'emp0001', 12);
insert into works_on values(1,'emp0002', 10);
insert into works_on values(2,'emp0001', 6);
insert into works_on values(3,'emp0004', 2);

insert into dependent values(‘emp0001’, ‘sunita’,’sister’);
insert into dependent values(‘emp0001’, ‘nita’,’mother’);
insert into dependent values(‘emp0002’, ‘kamal’,’brother’);
insert into dependent values('emp0004', 'krishna','father');


select * from employee;
select * from department;
select * from project;
select * from dependent;
select * from works_on;
```

1) employee

```
fnamemnamelnamessn        salary     ecitydno
-------------------------------------- ---------------------------- ---------------------------------
anita          m        sharma      emp0001    20000      mumbai101
juhi            r        verma       emp0002    15000      delhi          104
krupita        v        jetali        emp0003    20000      delhi          103
nita           g        patil          emp0004    10000      mumbai       101
anita          m        sharma      emp0005     20000     mumbai104
```

2) department

```
dnamednodlocation
---------------------------- ---------- ---------------------------------
IT                        101       mumbai
COMP                    102       mumbai
ETRX                    103       delhi
EXTC                    104       chennai
account                105       mumbai
```

4) project

```
pnoplocationpnamedno
---------- ------------------------------------ ------------------- ----------
1          mumbai                          website          101
2          chennai                        coding          101
3          mumbai                          testing          102
4          delhidocumentaion        103
```

5) dependents

```
ssndepname                   relation
------------------ ---------------------------- ----------------------------
emp0001nita                    mother
emp0001sunita                   sister
emp0002kamal                    brother
emp0004krishna                  father
```

6) woks_on

```
pnossnno_of_hrs
----------- ------------------- -----------
1         emp0001           12
1         emp0002           10
2         emp0001           6
3         emp0004           2
```

Results: (Queries printout with output as per the format)

INSERT

Code:

```
-- Insert rows into the schedules table
INSERT INTO schedules (route_id, bus_id, departure_time, arrival_time) VALUES
(1, 1, '2024-09-08 08:00:00', '2024-09-08 12:00:00'),
(2, 2, '2024-09-08 09:00:00', '2024-09-08 13:00:00'),
(3, 3, '2024-09-08 10:00:00', '2024-09-08 14:00:00'),
(4, 4, '2024-09-08 11:00:00', '2024-09-08 15:00:00'),
(5, 5, '2024-09-08 12:00:00', '2024-09-08 16:00:00');

-- View schedules table after insert
SELECT * FROM schedules;

-- Insert rows into the customers table
INSERT INTO customers (name, email, phone_number) VALUES
('Ritesh Jha', 'ritesh@example.com', '1234567890'),
('Anita Sharma', 'anita@example.com', '0987654321'),
('Nita Patil', 'nita@example.com', '1122334455');

-- View customers table after insert
SELECT * FROM customers;

-- Insert rows into the bookings table
INSERT INTO bookings (customer_id, schedule_id) VALUES
(1, 1),
(2, 2),
(3, 3);

-- View bookings table after insert
SELECT * FROM bookings;

-- Insert rows into the tickets table
INSERT INTO tickets (booking_id, seat_number) VALUES
(1, 'A1'),
(2, 'B1'),
```

(3, 'C1');

-- View tickets table after insert
SELECT * FROM tickets;

Output :

```
INSERT 0 5
 schedule_id | route_id | bus_id |    departure_time     |     arrival_time
-------------+----------+--------+---------------------+---------------------
           1 |        1 |      1 | 2024-09-08 08:00:00 | 2024-09-08 12:00:00
           2 |        2 |      2 | 2024-09-08 09:00:00 | 2024-09-08 13:00:00
           3 |        3 |      3 | 2024-09-08 10:00:00 | 2024-09-08 14:00:00
           4 |        4 |      4 | 2024-09-08 11:00:00 | 2024-09-08 15:00:00
           5 |        5 |      5 | 2024-09-08 12:00:00 | 2024-09-08 16:00:00
(5 rows)
```

```
INSERT 0 3
 customer_id |     name     |       email        | phone_number
-------------+--------------+--------------------+--------------
           1 | Ritesh Jha   | ritesh@example.com | 1234567890
           2 | Anita Sharma | anita@example.com  | 0987654321
           3 | Nita Patil   | nita@example.com   | 1122334455
(3 rows)
```

```
INSERT 0 3
 booking_id | customer_id | schedule_id |        booking_date
------------+-------------+-------------+----------------------------
          1 |           1 |           1 | 2024-09-08 10:59:59.324363
          2 |           2 |           2 | 2024-09-08 10:59:59.324363
          3 |           3 |           3 | 2024-09-08 10:59:59.324363
(3 rows)
```

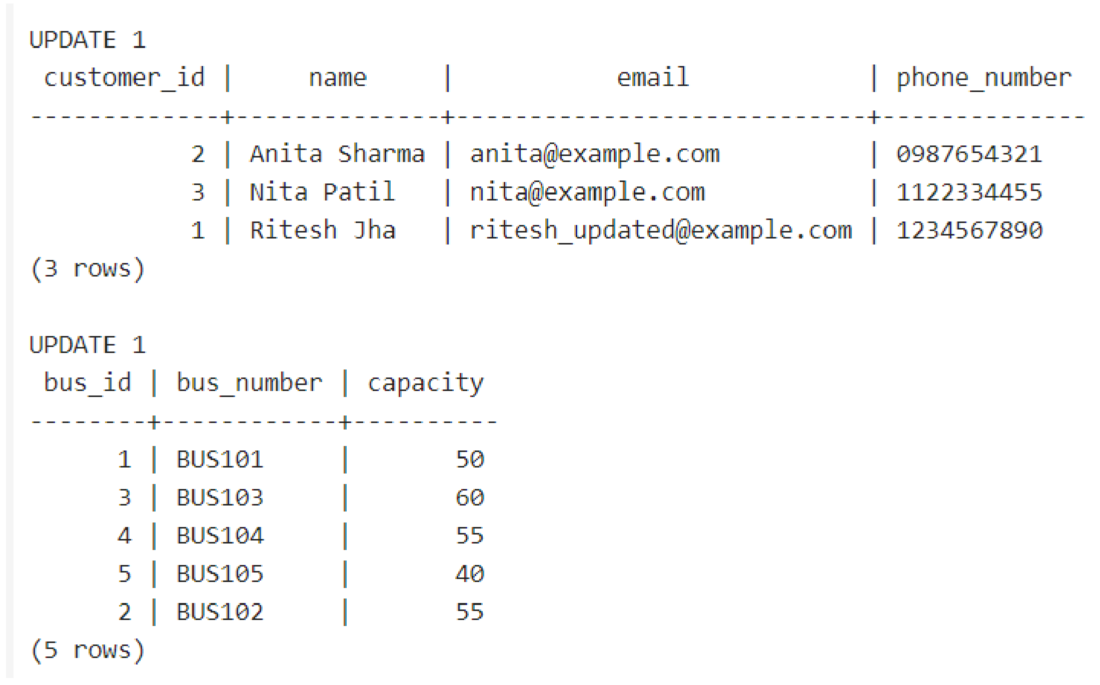
```
INSERT 0 3
 ticket_id | booking_id | seat_number
-----------+------------+-------------
         1 |          1 | A1
         2 |          2 | B1
         3 |          3 | C1
(3 rows)
```

UPDATE

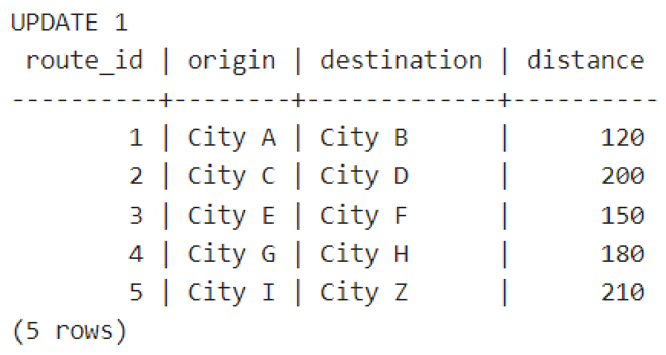
Code:

```
-- Update the email of a customer
UPDATE customers SET email = 'ritesh_updated@example.com' WHERE customer_id = 1;

-- View customers table after update
SELECT * FROM customers;

-- Increase the capacity of a bus
UPDATE buses SET capacity = capacity + 10 WHERE bus_id = 2;

-- View buses table after update
SELECT * FROM buses;

-- Change the destination of a route
UPDATE routes SET destination = 'City Z' WHERE route_id = 5;

-- View routes table after update
SELECT * FROM routes;

-- Update the phone number of a customer
UPDATE customers SET phone_number = '9988776655' WHERE customer_id = 3;

-- View customers table after update
SELECT * FROM customers;
```

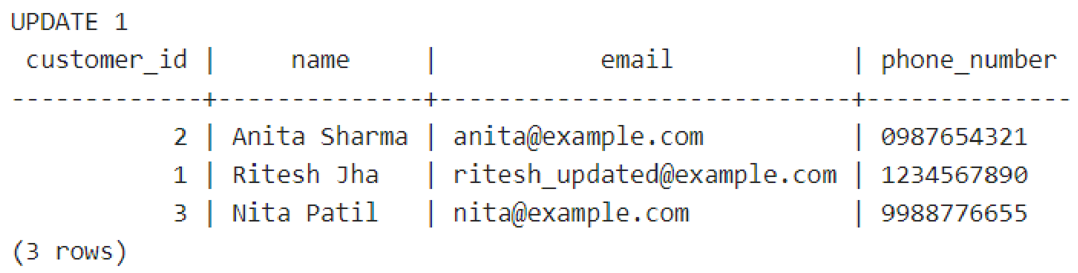
Output:



```
UPDATE 1
 customer_id |     name     |           email            | phone_number
-------------+--------------+----------------------------+--------------
           2 | Anita Sharma | anita@example.com          | 0987654321
           3 | Nita Patil   | nita@example.com           | 1122334455
           1 | Ritesh Jha   | ritesh_updated@example.com | 1234567890
(3 rows)

UPDATE 1
 bus_id | bus_number | capacity
--------+------------+----------
      1 | BUS101     |       50
      3 | BUS103     |       60
      4 | BUS104     |       55
      5 | BUS105     |       40
      2 | BUS102     |       55
(5 rows)
```



```
UPDATE 1
 route_id | origin | destination | distance
----------+--------+-------------+----------
        1 | City A | City B      |      120
        2 | City C | City D      |      200
        3 | City E | City F      |      150
        4 | City G | City H      |      180
        5 | City I | City Z      |      210
(5 rows)
```



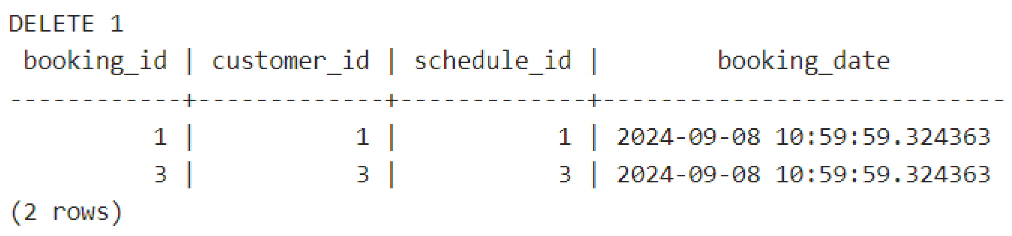
```
UPDATE 1
 customer_id |     name     |           email            | phone_number
-------------+--------------+----------------------------+--------------
           2 | Anita Sharma | anita@example.com          | 0987654321
           1 | Ritesh Jha   | ritesh_updated@example.com | 1234567890
           3 | Nita Patil   | nita@example.com           | 9988776655
(3 rows)
```

DELETE

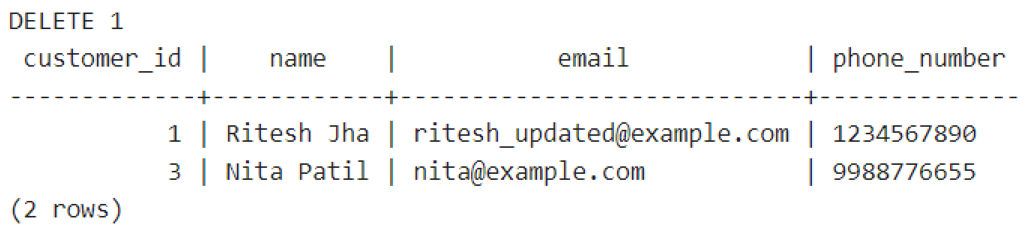
Code:

```
-- Delete a booking
DELETE FROM bookings WHERE booking_id = 2;

-- View bookings table after delete
SELECT * FROM bookings;

-- Remove a customer who has not made any bookings
DELETE FROM customers WHERE customer_id = 2;

-- View customers table after delete
SELECT * FROM customers;

-- Delete a bus with a specific bus number
DELETE FROM buses WHERE bus_number = 'BUS103';

-- View buses table after delete
SELECT * FROM buses;

-- Delete all routes with distance greater than 200
DELETE FROM routes WHERE distance > 200;

-- View routes table after delete
SELECT * FROM routes;
```

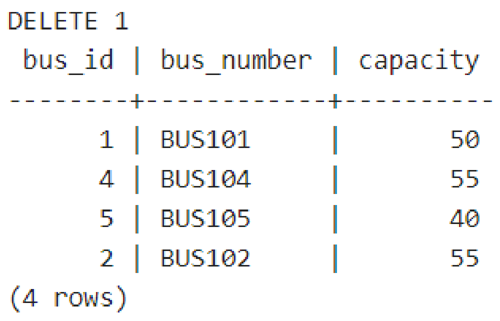
Output:



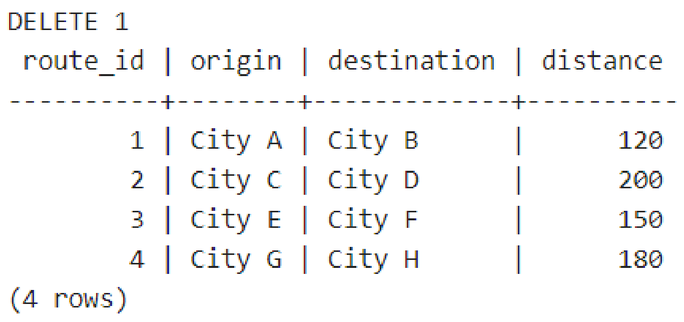
```
DELETE 1
 booking_id | customer_id | schedule_id |        booking_date
------------+-------------+-------------+----------------------------
          1 |           1 |           1 | 2024-09-08 10:59:59.324363
          3 |           3 |           3 | 2024-09-08 10:59:59.324363
(2 rows)
```



```
DELETE 1
 customer_id |    name    |           email            | phone_number
-------------+------------+----------------------------+--------------
           1 | Ritesh Jha | ritesh_updated@example.com | 1234567890
           3 | Nita Patil | nita@example.com           | 9988776655
(2 rows)
```

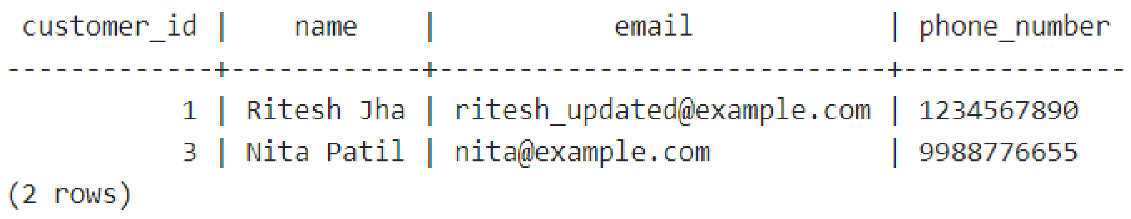


```
DELETE 1
 bus_id | bus_number | capacity
--------+------------+----------
      1 | BUS101     |       50
      4 | BUS104     |       55
      5 | BUS105     |       40
      2 | BUS102     |       55
(4 rows)
```



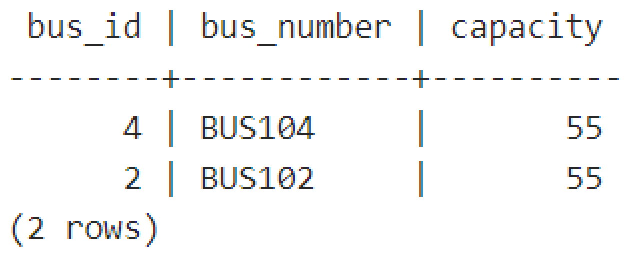
```
DELETE 1
 route_id | origin | destination | distance
----------+--------+-------------+----------
        1 | City A | City B      |      120
        2 | City C | City D      |      200
        3 | City E | City F      |      150
        4 | City G | City H      |      180
(4 rows)
```

1. **Write 10 queries using ‘from’ and ‘where’ clause.**
   -- Select all customers
   SELECT * FROM customers;



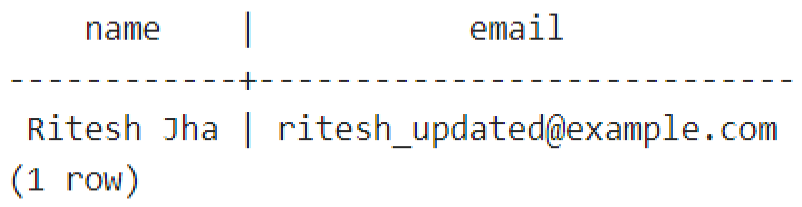
```
 customer_id |    name    |           email            | phone_number
-------------+------------+----------------------------+--------------
           1 | Ritesh Jha | ritesh_updated@example.com | 1234567890
           3 | Nita Patil | nita@example.com           | 9988776655
(2 rows)
```

-- Select all buses with a capacity greater than 50
SELECT * FROM buses WHERE capacity > 50;



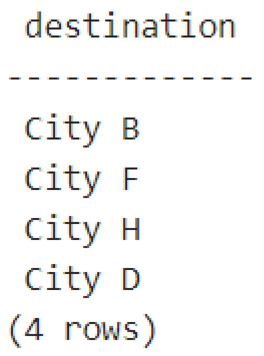
```
 bus_id | bus_number | capacity
--------+------------+----------
      4 | BUS104     |       55
      2 | BUS102     |       55
(2 rows)
```

-- Select the name and email of customers who have booked a specific schedule
SELECT name, email FROM customers WHERE customer_id IN (SELECT customer_id FROM bookings WHERE schedule_id = 1);



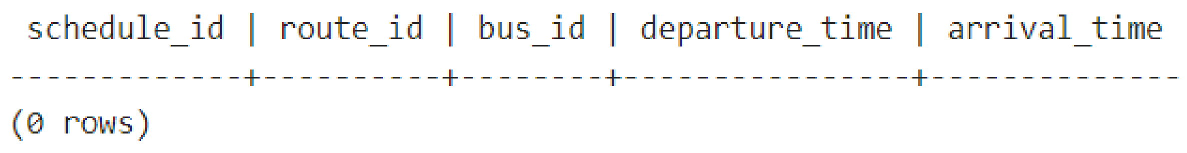
```
    name     |            email
-------------+----------------------------
 Ritesh Jha | ritesh_updated@example.com
(1 row)
```

-- Select distinct destinations from the routes table
SELECT DISTINCT destination FROM routes;



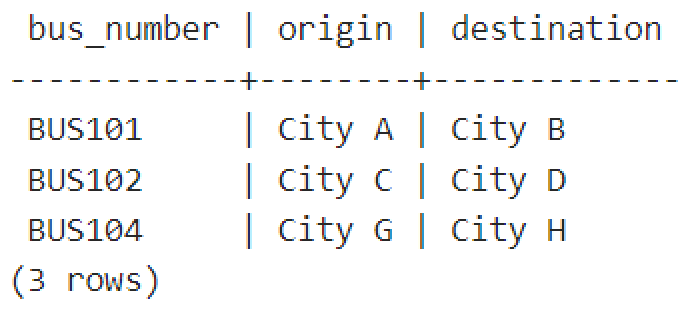
```
 destination
-------------
 City B
 City F
 City H
 City D
(4 rows)
```

-- Select schedules where the departure time is before a specific time
SELECT * FROM schedules WHERE departure_time < '2024-09-06 10:00:00';



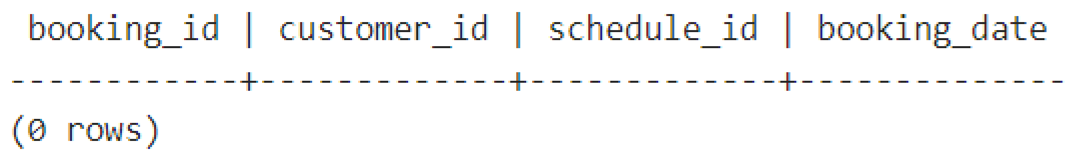
```
 schedule_id | route_id | bus_id | departure_time | arrival_time
-------------+----------+--------+----------------+--------------
(0 rows)
```

-- Select the bus number and route details for all schedules
SELECT bus_number, origin, destination FROM schedules
JOIN buses ON schedules.bus_id = buses.bus_id
JOIN routes ON schedules.route_id = routes.route_id;



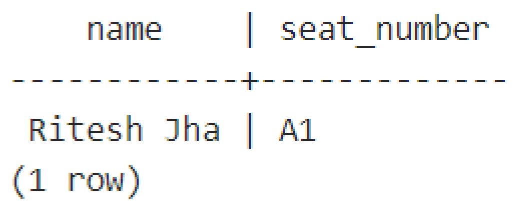
```
 bus_number | origin | destination
------------+--------+-------------
 BUS101     | City A | City B
 BUS102     | City C | City D
 BUS104     | City G | City H
(3 rows)
```

-- Select bookings made on a specific date
SELECT * FROM bookings WHERE booking_date::date = '2024-09-06';



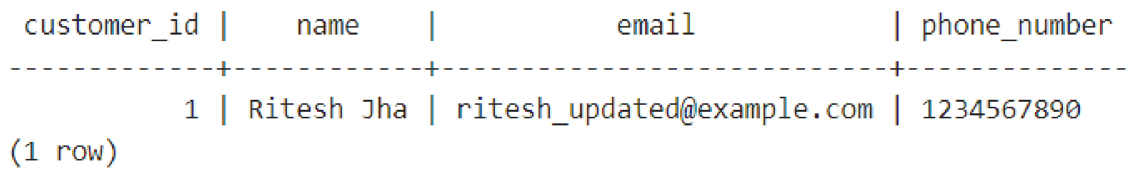
```
 booking_id | customer_id | schedule_id | booking_date
------------+-------------+-------------+--------------
(0 rows)
```

-- Select the customer name and their booked seat numbers
SELECT name, seat_number FROM customers
JOIN bookings ON customers.customer_id = bookings.customer_id
JOIN tickets ON bookings.booking_id = tickets.booking_id;



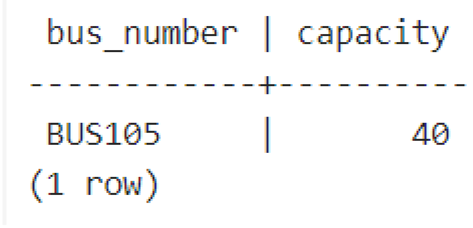
```
    name    | seat_number
------------+-------------
 Ritesh Jha | A1
(1 row)
```

-- Select customers with phone numbers starting with '123'
SELECT * FROM customers WHERE phone_number LIKE '123%';



```
 customer_id |    name    |           email            | phone_number
-------------+------------+----------------------------+--------------
           1 | Ritesh Jha | ritesh_updated@example.com | 1234567890
(1 row)
```

-- Select the bus number and capacity for all buses with a capacity less than 50
SELECT bus_number, capacity FROM buses WHERE capacity < 50;



```
 bus_number | capacity
------------+----------
 BUS105     |       40
(1 row)
```

Example:

1) **To extract the name and ssn of all the employees:**
Select fname, mname, lname, ssn from employee;

```
fnamemnamelnamessn
-------------------------------------- -------------------------------------- ----------------------------
anitasharmam                     emp0001
juhiverma                        r                    emp0002
krupitajetali                     v                    emp0003
nitapatil                      g                    emp0004
anitasharma                       m                    emp0005
```

2) **To select names and city of the employees earning salary more then 10000:**

Select fname, mname, lname, ecity from the employee where salary>10000;

```
fnamemnamelname              ecity
------------------------------------- ----------------------- ------------------------------------
anitasharmam                   mumbai
juhivermar                    delhi
krupitajetaliv                  delhi
anitasharma m                  mumbai
```

3) **TO get the details of the cities of the employees in our company:**

select distinct ecity from employee;

```
ecity
------------
delhi
mumbai
```

4) **To find the name of the department located in Mumbai and with department number 101:**

select dname from department where dlocation='Mumbai' and dno=101;

```
dname
--------------
```

5) **To delete all dependent whose relation is mother with employee:**

delete form dependent where relation='mother';

```
ssndepname                  relation
------------------- ----------------------------- -----------------------------
emp0001sunita                  sister
emp0002kamal                   brother
emp0004krishna                 father
```

6) **Update relation employee to increment salary of all employees working in Department 101 by Rs. 10000:**

update employee set salary=salary+10000 where dno=101;

```
fnamemnamelnamessn         salary      ecitydno
------------------------------------- ----------------------------- ---------------------------------
anita          m         sharma       emp0001      30000      mumbai101
juhi            r         verma         emp0002     15000      delhi             104
krupita        v         jetali          emp0003      20000      delhi             103
nita           g         patil emp0004       20000      mumbai        101
anita          m         sharma       emp0005       20000     mumbai104
```

Outcomes:

CO2 Apply data models to real world scenarios.

Questions:

Q1 Explain various data types used in SQL

In SQL, data types define the kind of data that can be stored in a column of a table. Here's an overview of some common data types:

1. **Numeric Data Types**:
   - **INT/INTEGER**: Used for whole numbers (e.g., 1, 2, 3). It can hold both positive and negative values.
   - **FLOAT/REAL**: Used for floating-point numbers, i.e., numbers with decimals (e.g., 3.14, 2.718).
   - **DECIMAL/NUMERIC**: Used for fixed-point numbers where you can define the precision (number of digits) and scale (number of digits after the decimal point).
2. **Character Data Types**:
   - **CHAR(size)**: Fixed-length string. If you specify CHAR(10), it will always use 10 characters, padding with spaces if necessary.
   - **VARCHAR(size)**: Variable-length string. Unlike CHAR, it only uses as much space as needed, up to the specified size.
   - **TEXT**: Used for large amounts of text. Unlike VARCHAR, the size limit is much larger.
3. **Date and Time Data Types**:
   - **DATE**: Stores a date in the format 'YYYY-MM-DD'.
   - **TIME**: Stores a time in the format 'HH:MM'.
   - **DATETIME**: Combines both date and time in the format 'YYYY-MM-DD HH:MM'.
   - **TIMESTAMP**: Similar to DATETIME, but it includes time zone information.
4. **Binary Data Types**:
   - **BINARY(size)**: Fixed-length binary data.
   - **VARBINARY(size)**: Variable-length binary data.
   - **BLOB**: Used for large binary data, like images or files.
5. **Boolean Data Types**:
   - **BOOLEAN**: Stores TRUE or FALSE values. In some databases, this is represented as 1 (TRUE) and 0 (FALSE).

Q2 what is outer JOIN and why it is used? Explain its type with example

Outer JOIN is used in SQL to combine rows from two or more tables based on a related column, including rows that do not have matching values. Unlike an INNER JOIN, which only returns rows with matching values, an OUTER JOIN returns all rows from one or both tables, even if there is no match.

LEFT OUTER JOIN (or LEFT JOIN):

Description: Returns all rows from the left table and the matched rows from the right table. If no match is found, NULL values are returned for columns from the right table.

Example:
sql
Copy code

```sql
SELECT Employees.Name, Orders.OrderID
FROM Employees
LEFT JOIN Orders ON Employees.EmployeeID =
Orders.EmployeeID;
```

This query returns all employees, including those who have not placed any orders. For employees without orders, the `OrderID` will be NULL.

- **RIGHT OUTER JOIN (or RIGHT JOIN)**:

  Description: Returns all rows from the right table and the matched rows from the left table. If no match is found, NULL values are returned for columns from the left table.

  Example:
  sql
  Copy code

  ```sql
  SELECT Employees.Name, Orders.OrderID
  FROM Employees
  RIGHT JOIN Orders ON Employees.EmployeeID =
  Orders.EmployeeID;
  ```

  This query returns all orders, including those that are not associated with any employee. For such orders, the `Name` will be NULL.

- **FULL OUTER JOIN (or FULL JOIN)**:

  Description: Returns all rows when there is a match in either the left or right table. If there is no match, the result is NULL on the side where no match is found.

Example:
sql
Copy code
```
SELECT Employees.Name, Orders.OrderID
FROM Employees
FULL OUTER JOIN Orders ON Employees.EmployeeID =
Orders.EmployeeID;
```

This query returns all employees and all orders, including employees without orders and orders without associated employees. Where there is no match, NULLs are filled in.

Usage:

- **OUTER JOINs** are particularly useful when you need to include records that do not have a match in the related table, such as generating reports that require all entities to be listed, even if they lack corresponding data in another table.

Conclusion: (Conclusion to be based on the objectives and outcomes achieved)

From the above experiment, I learned how to create and manage tables in a Postgres database, including how to insert, update, and delete data. I also practiced writing SQL queries to retrieve specific information from the tables. This helped me understand how different tables can be linked using foreign keys and how to use joins to combine data from multiple tables. Overall, I got a better grasp of how to work with databases in a structured and efficient way.

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of faculty in-charge with date

References:

Books:

1. Elmasri and Navathe, "Fundamentals of Database Systems", 6th Edition, Pearson Education
2. Korth, Slberchatz,Sudarshan, :"Database System Concepts", 6th Edition, McGraw – Hill.

WebSite:
1. http://www.tutorialspoint.com/postgresql/

2. http://sage.virtual-labs.ac.in/home/pub/21/