**Practical No. 7**

**Aim**: Data retrieval from the designed database. e.g. JOINS

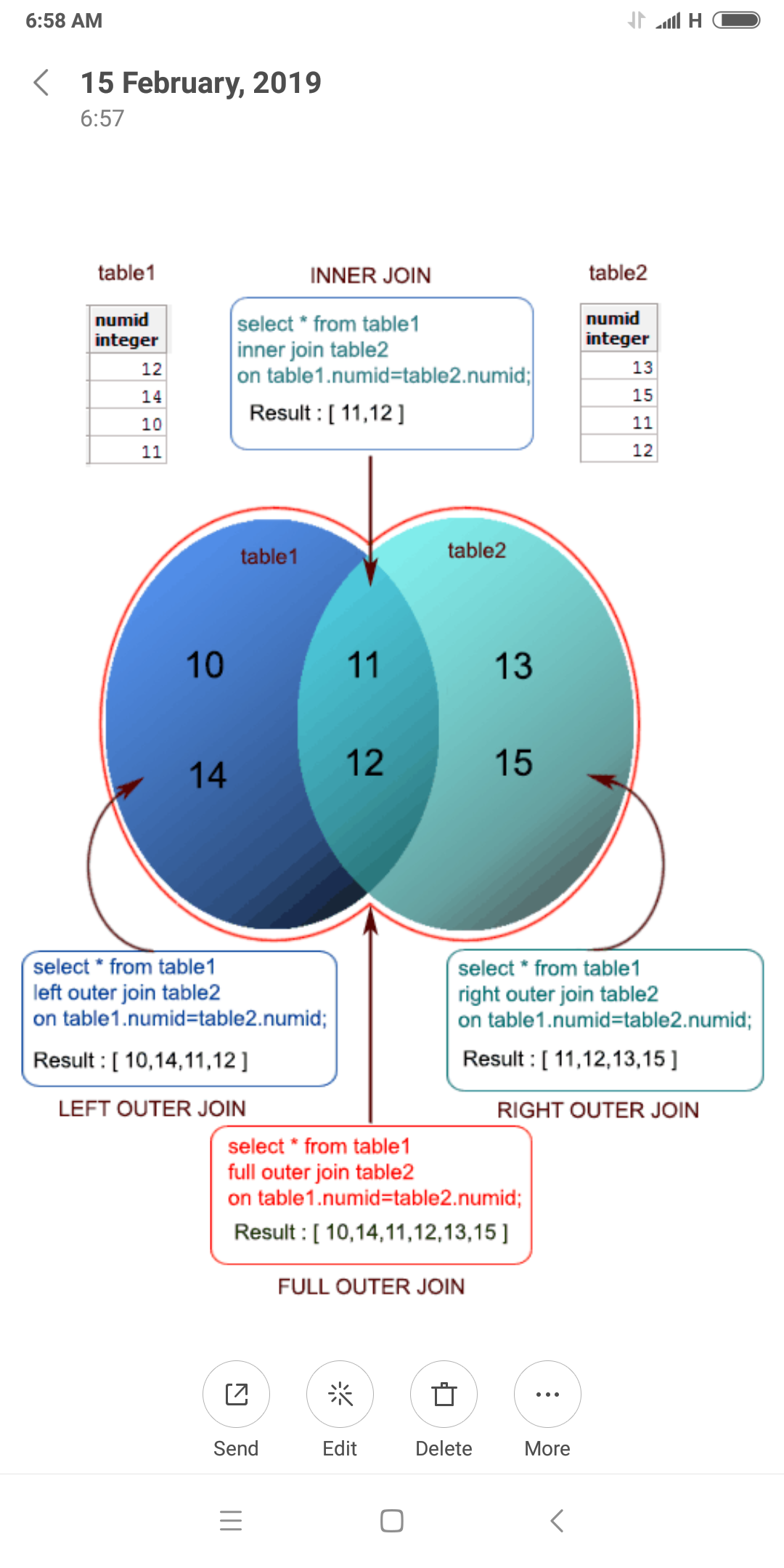
***JOINS in SQL***

The main concept which is focusing on a join is that, two or more data sets, when joined, combined their columns into a new set of rows, including each of the columns requested from each of the data sets. All joins are standing on the foundation of Cartesian product. The Cartesian product is the set of all possible combinations between two data sets. A join creates a set of rows in a temporary table and works on two or more tables, and each table should at least one common field and must maintain a relation between the common fields. Join keeps the structure unchanged of the base tables.

Types of PostgreSQL JOIN

* Cross Join
* Inner Join
* Left Outer Join
* Right Outer Join
* Full Outer Join

**Pictorial Presentation of PostgreSQL Joins**



Suppose we have two tables :

postgres=# select \* from supplier;

id | name

----+------------

0 | Jems

1 | Pat

2 | Kane

(3 rows)

postgres=# select \* from orders;

id | ord\_value

----+-----------

1 | 5000

2 | 8000

3 | 10000

(3 rows)

**CROSS JOIN**

The simplest type of join is a cross join. The Cross Join creates a cartesian product between two sets of data.

postgres=# select \* from supplier, orders;

id | name | id | ord\_value

----+------------+----+-----------

0 | Jems| 1 | 5000

1 | Pat | 1 | 5000

2 | Kane | 1 | 5000

0 | Jems| 2 | 8000

1 | Pat | 2 | 8000

2 | Kane | 2 | 8000

0 | Jems| 3 | 10000

1 | Pat | 3 | 10000

2 | Kane | 3 | 10000

(9 rows)

The above result shows that the first two columns are all the fields of the first table’s supplier, followed by all the fields of the second table, orders. Here from the above result, it seems that it takes the supplier table and combines all its rows with the first row of orders, then it takes the supplier table again and combines with the second row of orders table and finally does the same job for the third row of orders to produce 3 \* 3 rows.

Here in the below statement shows that the same set of results can be produced by using the CROSS JOIN terminology. Here is the code below.

postgres=# select \* from supplier cross join orders;

id | name | id | ord\_value

----+------------+----+-----------

0 | Jems| 1 | 5000

1 | Pat | 1 | 5000

2 | Kane | 1 | 5000

0 | Jems| 2 | 8000

1 | Pat | 2 | 8000

2 | Kane | 2 | 8000

0 | Jems| 3 | 10000

1 | Pat | 3 | 10000

2 | Kane | 3 | 10000

(9 rows)

**INNER JOIN**

If we want to restrict the set of combinations based on some join criteria to determine which rows from both participating tables are considered to return on a match between the columns. Suppose we want to match up the supplier and order Ids. Here is the statement below.

postgres=# select \* from supplier join orders on supplier.id=orders.id;

id | name | id | ord\_value

----+------------+----+-----------

1 | Pat | 1 | 5000

2 | Kane | 2 | 8000

(2 rows)

or the query can be written like below.

postgres=# select \* from supplier inner join orders on supplier.id=orders.id;

id | name | id | ord\_value

----+------------+----+-----------

1 | Pat | 1 | 5000

2 | Kane | 2 | 8000

(2 rows)

The above output shows that only those rows have in output who matches the criteria in both the associated table, that is why we obtain only 2 rows (1,'Pat') and (2,'Kane'). The criteria are boolean criteria and it must return true or false for each row combination.

**NATURAL JOIN**

Sometimes we use the same field name for the same entity types across different tables. A NATURAL join is a special type of INNER join where we ask PostgreSQL to match up on all columns that have the same name. Here is the query below.

postgres=# select \* from supplier natural join orders;

id | name | ord\_value

----+------------+-----------

1 | Pat | 5000

2 | Kane | 8000

(2 rows)

In the above example, the 'id'column appear only once, because this column is common in both the tables. Here, PostgreSQL implicitly does an ***INNER JOIN*** on that column.

**LEFT JOIN or LEFT OUTER JOIN**

In LEFT JOIN when joining table supplier with table orders PostgreSQL first does a "normal" inner join. Then it starts to scan if there are any rows from supplier table that are not in the result set. If so, it adds in those rows placing NULLs for all the fields of orders. Thus, make sure that each row of the supplier will appear at least once in the result set. Here is the example below.

postgres=# select \* from supplier left join orders on supplier.id=orders.id;

or the query can be written like below.

postgres=# select \* from supplier left outer join orders on supplier.id=orders.id;

id | name | id | ord\_value

----+------------+----+-----------

0 | Jems | |

1 | Pat | 1 | 5000

2 | Kane | 2 | 8000

(3 rows)

The above example shows that the row highlight in blue color, the id 0 in supplier table does not match with the table orders. so, for all the fields of orders it replaces NULL, here in the above output, the blank fields are NULL value.

**RIGHT JOIN or RIGHT OUTER JOIN**

In RIGHT JOIN when joining table supplier with table orders PostgreSQL first does a "normal" inner join. Then it starts to scan if there are any rows from orders table that are not in the result set. If so, it adds in those rows placing NULLs for all the fields of the supplier. Thus, make sure that each row of orders will appear at least once in the result set. Here is the example below.

postgres=# select \* from supplier right join orders on supplier.id=orders.id;

or the query can be written like below.

postgres=# select \* from supplier right outer join orders on supplier.id=orders.id;

id | name | id | ord\_value

----+------------+----+-----------

1 | Pat | 1 | 5000

2 | Kane | 2 | 8000

| | 3 | 10000

(3 rows)

The above example shows that the row highlight in blue color, the id 3 in orders table does not match with the table supplier, so, for all the fields of supplier, it replaces NULL, here in the above output the blank fields are NULL value.

**FULL JOIN or FULL OUTER JOIN**

The FULL OUTER JOIN is a combination of a left and a right outer join. Firstly it does an inner join and adds in any missing rows from both  the table associated with this statement, here supplier and orders.

postgres=# select \* from supplier full join orders on supplier.id=orders.id;

or the query can be written like below.

postgres=# select \* from supplier full outer join orders on supplier.id=orders.id;

id | name | id | ord\_value

----+------------+----+-----------

0 | Jems | |

1 | Pat | 1 | 5000

2 | Kane | 2 | 8000

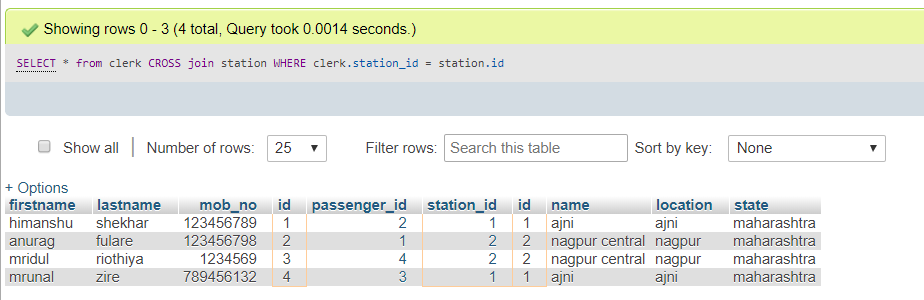
| | 3 | 10000

(4 rows)

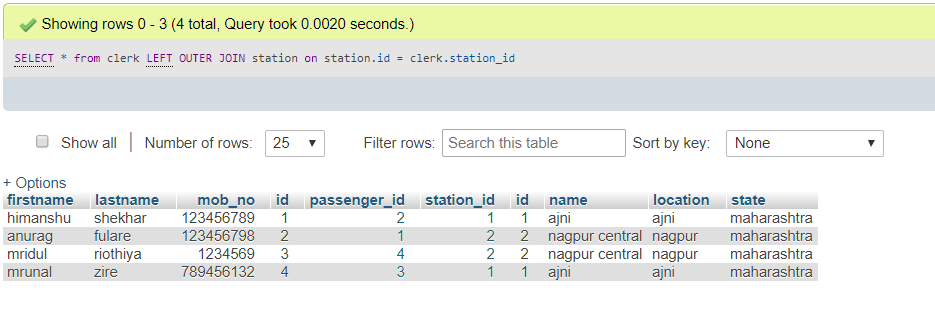
The above example shows that the row highlight in blue color, the id 0 and 3 in supplier and orders table does not match with the table orders or supplier respectively, so, for all the fields of orders and supplier it replaces NULL, here in the above output, the blank fields are NULL value.

Execution on our project

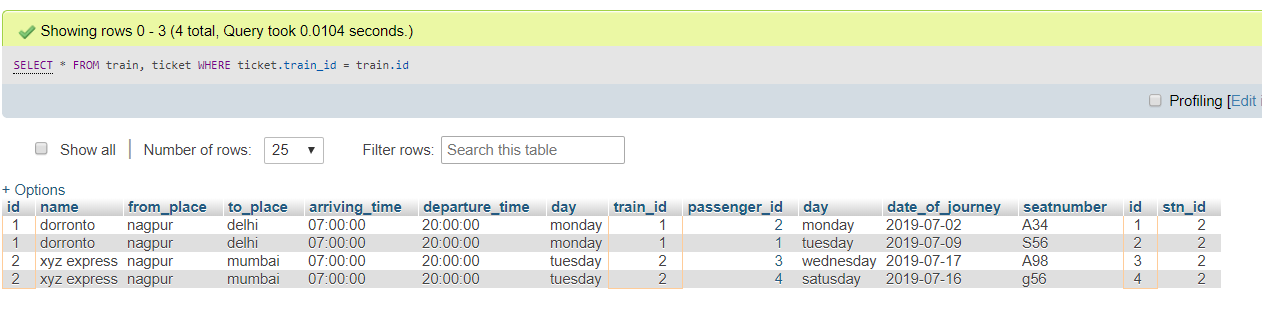
Q.1 Show all the details of clerk and their corresponding sitting station?

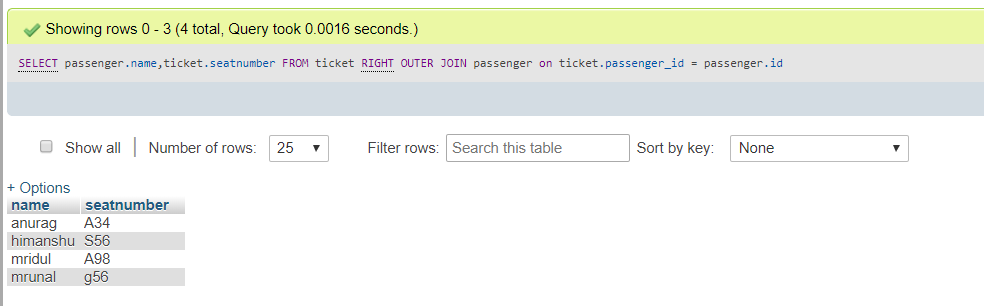


Q2. Show all the details of clerk whose sitting at any station?

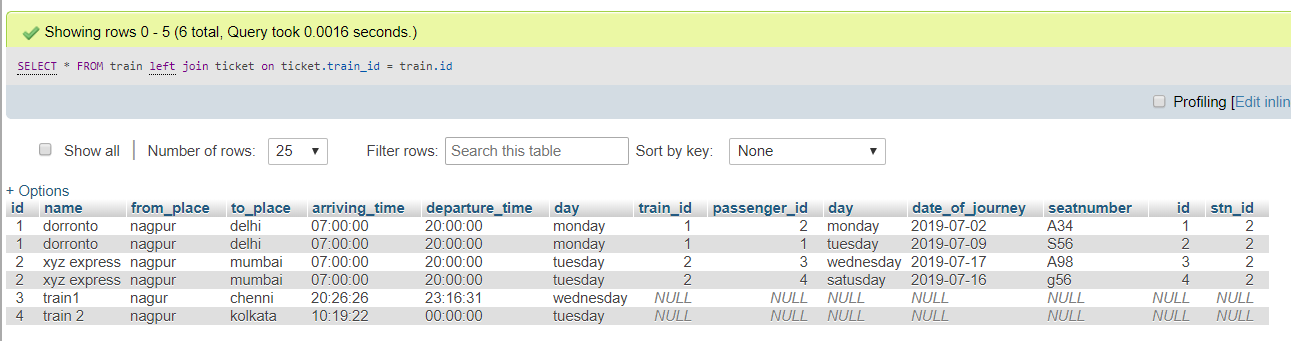


Q3. Show the details of trains and the corresponding booked tickets on them?

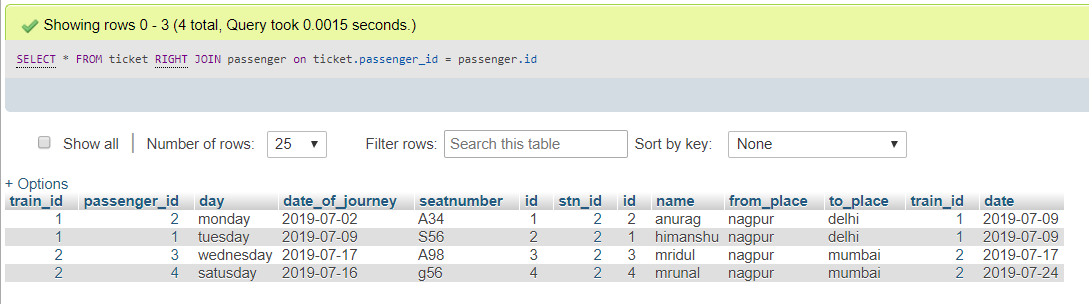


Q4. Show the passenger name and the corresponding seatnumbers.

Q5. Show all the details of tickets and their boarding trains?



Q6. Show all the details of passenger with their ticket details?



**Conclusion:**

Thus we have created various types of Joins(cross join, inner/outer left/right join full join) on various tables of our project.