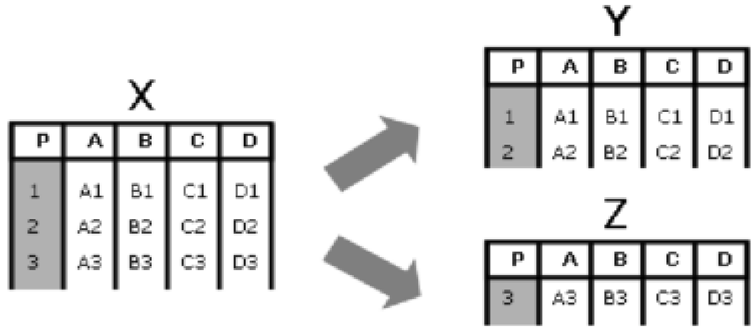
Name:	Shubham Golwal
UID no.	2020300015
Experiment No.	03

AIM:	Horizontal fragmentation
Program 1	
Problem Statement :	Design a distributed database system by applying the concept of horizontal fragmentation.
Theory :	**What is Fragmentation?** Fragmentation in ADBMS (Advanced Database Management Systems) refers to the process of dividing a large database into smaller and more manageable parts, called fragments. This is done to improve the performance, scalability, and availability of the database by distributing the data across multiple servers, disks, or storage devices. This can also help in reducing the size of individual fragments and improve the access time for specific data subsets, making it easier to manage and maintain the database. There are two main types of fragmentation in ADBMS: 1. Horizontal Fragmentation 2. Vertical Fragmentation **Horizontal Fragmentation** Horizontal fragmentation is a type of fragmentation in ADBMS (Advanced Database Management Systems) where a database table is divided into multiple smaller tables with the same schema but containing different rows of data. This type of fragmentation is also known as row-level fragmentation, as it splits the data horizontally across different tables. The purpose of horizontal fragmentation is to distribute the data across multiple servers or storage devices, improving scalability, query performance, and availability of the database. This can also help in reducing the size of individual fragments and improve the access time for specific data subsets, making it easier to manage and maintain the database.



Correctness rules checking the correctness criteria of horizontal fragmentation

Completeness: If relation R is decomposed into fragments R1,R2,. Rn each data item that can be found in R can also be found in one or more Ri’s.

Reconstruction: If relation R is decomposed into fragments R1,R2,. Rn , it should be possible to define relational operator delta such that R=delta(Ri) for all Ri belongs to Fr.

Disjointness : If relation R is horizontally decomposed into fragments R1,R2... Rn and data item di is in Rj it is not in any other fragment Rk (k not equal to 1)

Advantages of fragmentation

Before we discuss fragmentation in detail, we list four reasons for fragmenting a relation

Usage

In general, applications work with views rather than entire relations. Therefore, for data distribution, it seems appropriate to work with subsets of relation as the unit of distribution.

Efficiency

Data is stored close to where it is most frequently used. In addition, data that is ‚not needed by’ local applications is not stored.

Parallelism

With fragments as the unit of distribution, a transaction can be divided into several sub queries that operate on fragments. This should increase the degree of concurrency, or parallelism, in the system, thereby allowing transactions that can do so safely to execute in parallel.

	Security Data not required by local applications is not stored, and consequently not available to unauthorized users. **Disadvantages of fragmentation** Fragmentation has two primary disadvantages, which we have mentioned previously: **Performance** The performance of global application that requires data from several fragments located at different sites may be slower. **Integrity** Integrity control may be more difficult if data and functional dependencies are fragmented and located at different sites.
Queries:	**-- Table: public.passengers** **-- DROP TABLE IF EXISTS public.passengers;** **CREATE TABLE IF NOT EXISTS public.passengers** **(** **passenger_id integer NOT NULL,** **first_name character varying(255) COLLATE pg_catalog."default" NOT NULL,** **last_name character varying(255) COLLATE pg_catalog."default" NOT NULL,** **email character varying(255) COLLATE pg_catalog."default" NOT NULL,** **ticket_fare numeric(10,2) NOT NULL,** **CONSTRAINT passengers_pkey PRIMARY KEY (passenger_id)** **)** **TABLESPACE pg_default;** **ALTER TABLE IF EXISTS public.passengers** **OWNER to postgres;** **select * from passengers;** **Create table pass3 as**

```
select * from passengers
where ticket_fare >= 3000;

select * from pass3;

Create table pass4 as
select * from passengers
where ticket_fare < 3000;

select * from pass4;

union
select first_name, last_name, ticket_fare from pass3 union all select
first_name, last_name, ticket_fare from pass4;

find
select email from pass3 where ticket_fare = 2500;

select first_name, last_name, ticket_fare from pass3 where
passenger_id = 1 union all select first_name, last_name, ticket_fare
from pass4 where passenger_id = 1;

select first_name, last_name, ticket_fare, email from pass3 where
passenger_id = 1 union all select first_name, last_name, ticket_fare,
email from pass4 where passenger_id = 1;

sum
select sum(ticket_fare)
from pass4
where ticket_fare < 2900;

sum -> null
select sum(ticket_fare)
from pass4
where ticket_fare < 1000;

select count(passenger_id), email
from pass3
group by email;
```

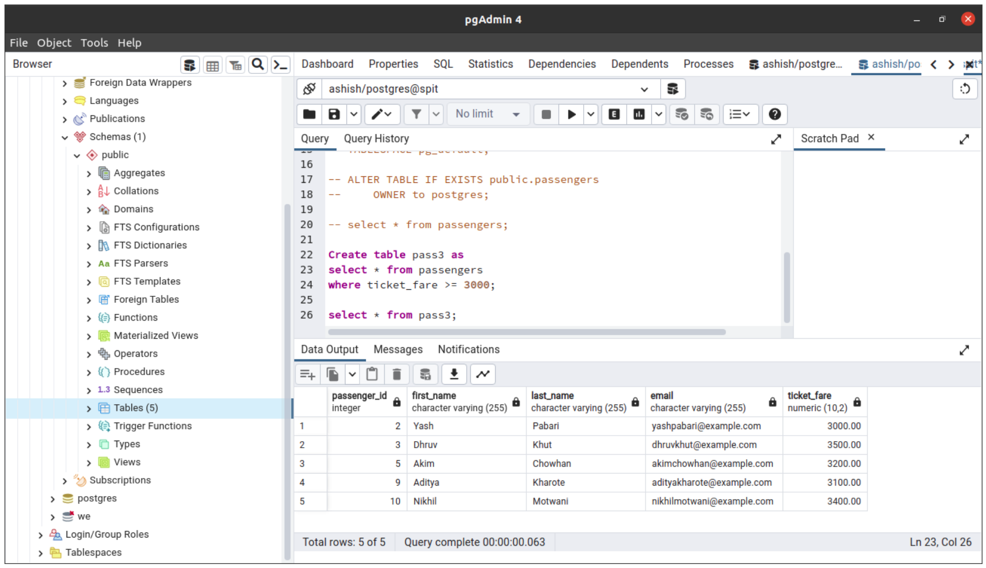
select * from pass3 order by ticket_fare desc limit 3;

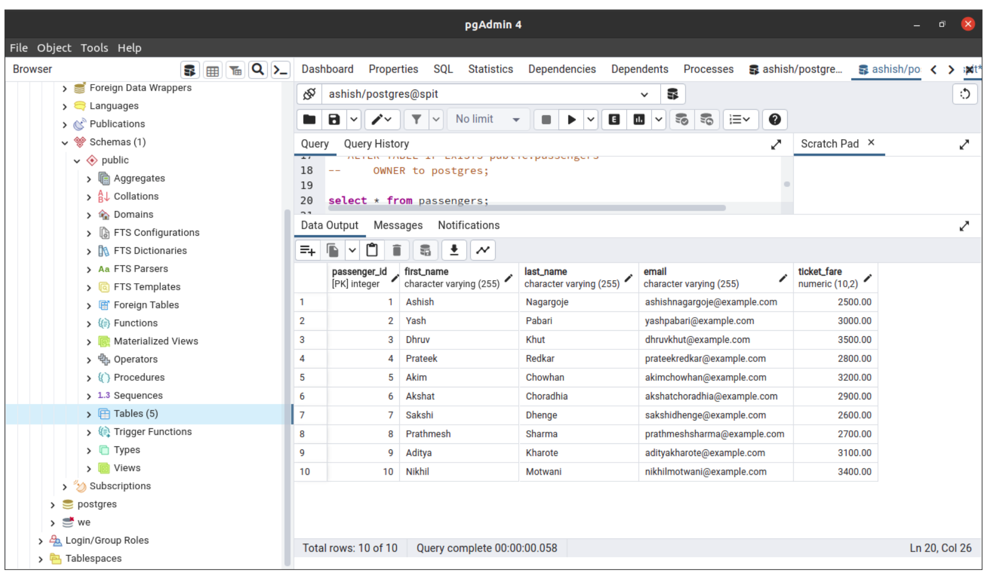
select * from pass3 union all select * from pass4;

select * from pass3 intersect select * from pass4;

Output

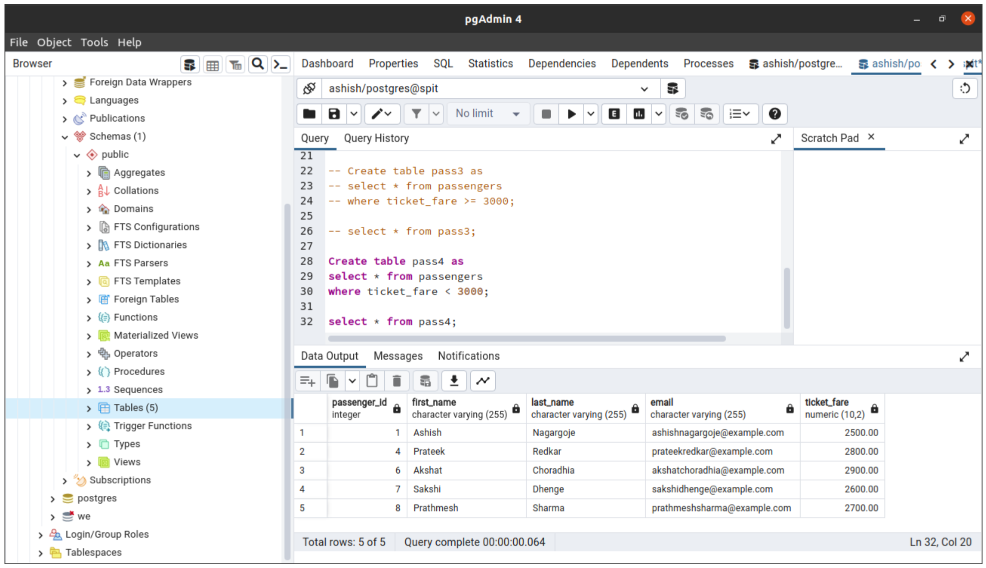
Create a global conceptual schema passengers(passenger_id, firdt_name, last_name, Email, ticket_fare) and insert 10 records.



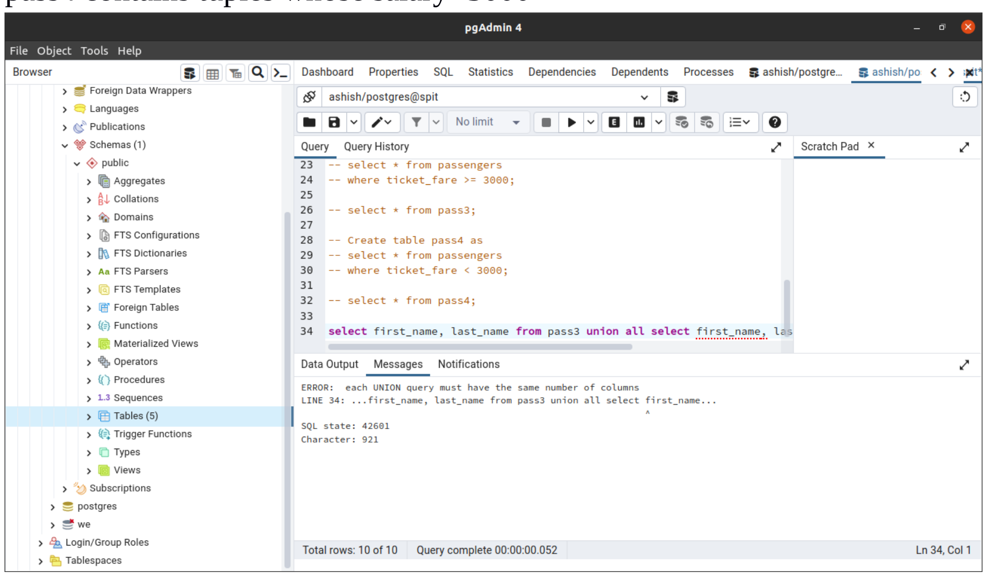


1. Divide the Passengers into two horizontal fragments pass3 and pass4.

pass3 contains tuples whose
salary >=3000

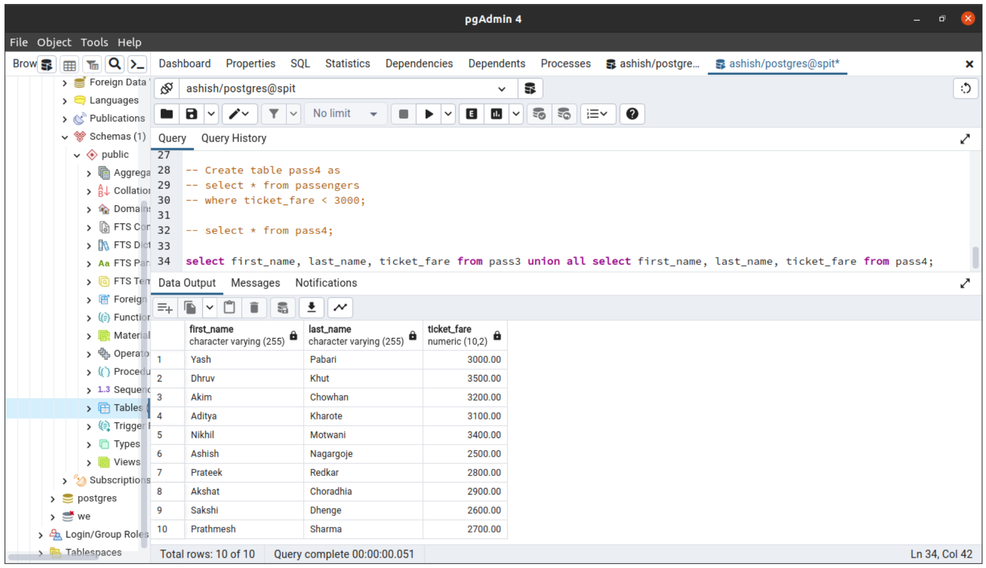


pass4 contains tuples whose salary<3000

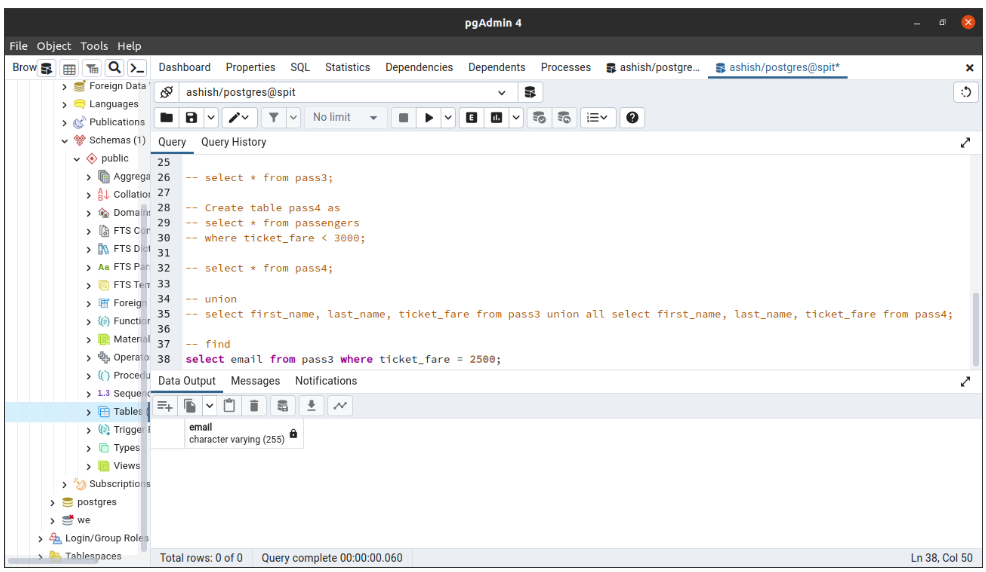


Execute following queries on horizontal fragments:

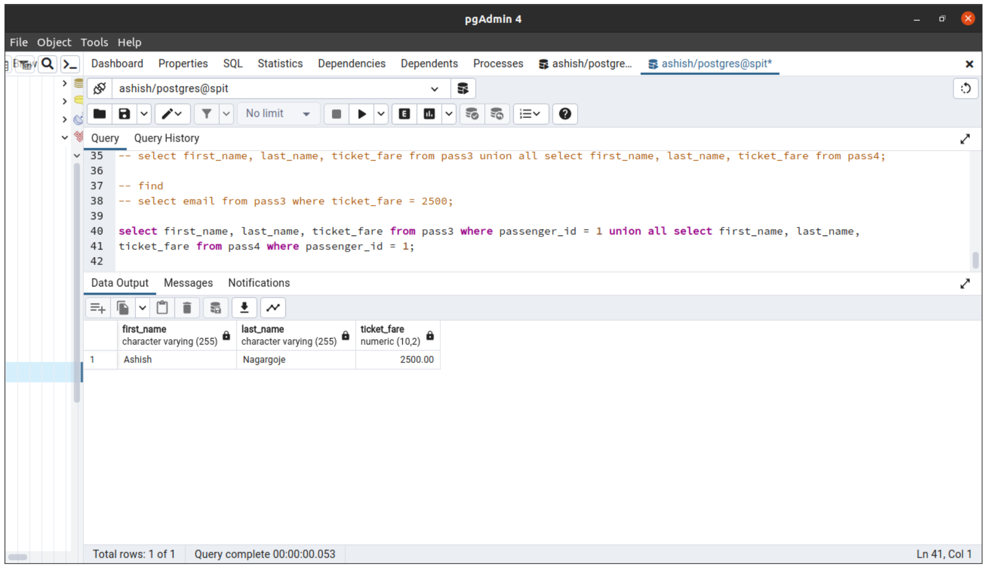
a. Find ticket price of all Passengers



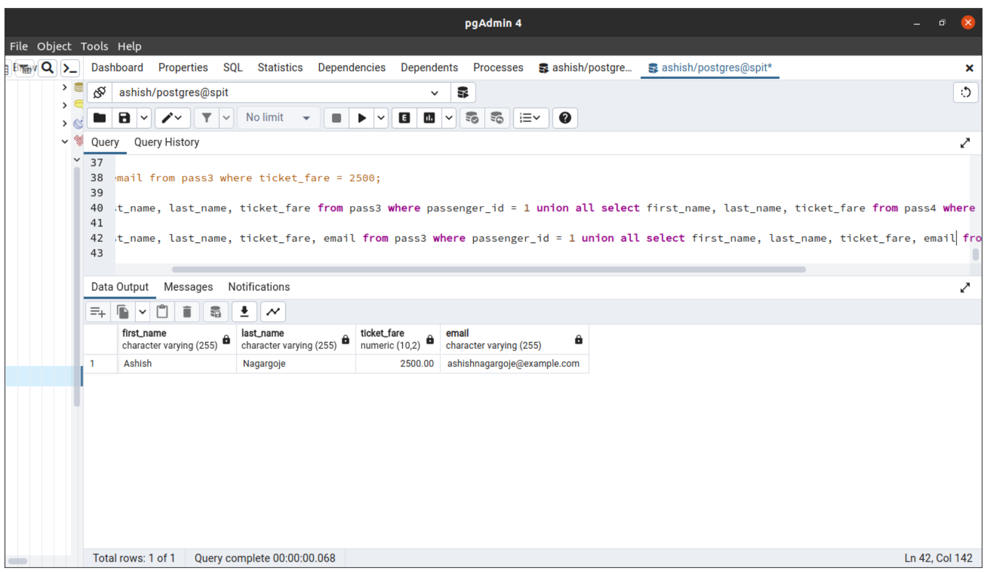
b. Find the email of all Passengers where Salary=2500



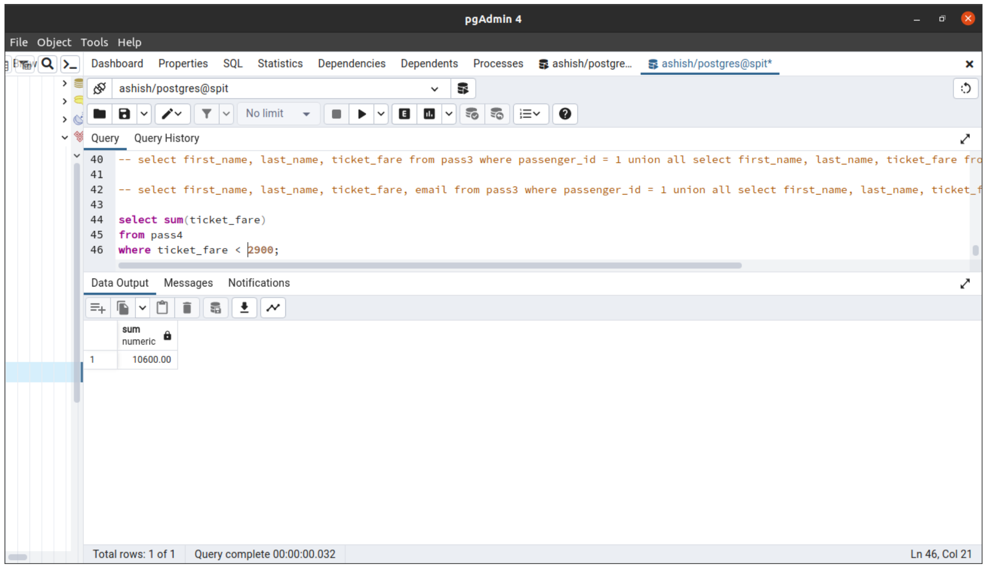
c. Find first_name, last_name and ticket price where passenger id is known



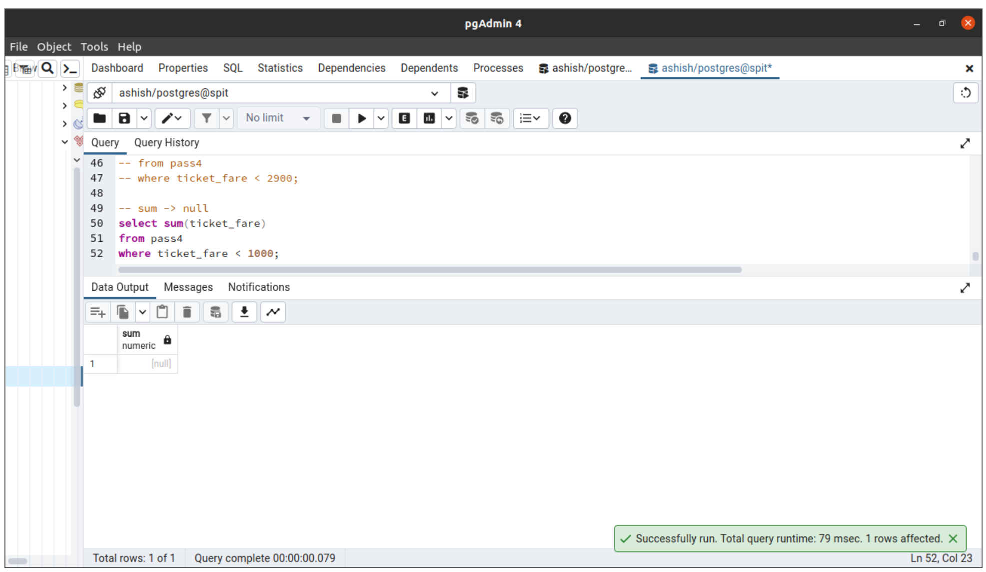
d. Find the first_name, last_name, email and ticket price where passenger id is known



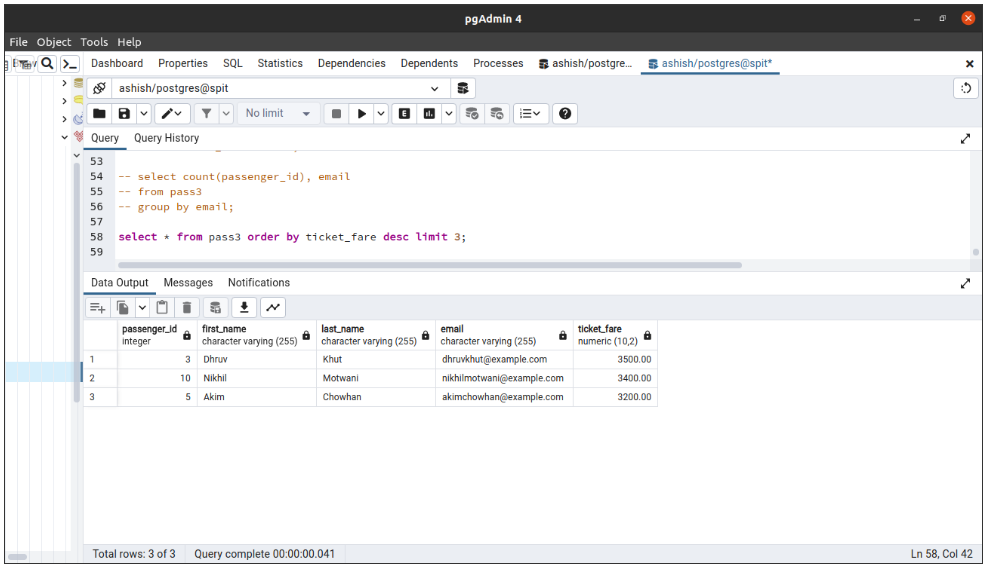
e. Find the sum of ticket prices of passengers where salary < 2500 (pass3)



pass4 (We will get null value as fragment pass4 contains data whose salary is less than 1000)

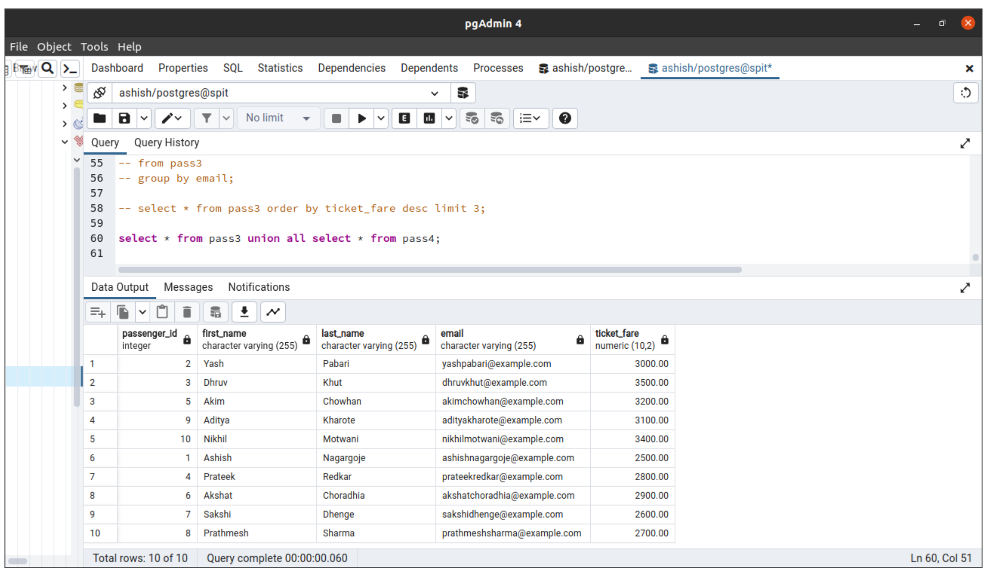


g. Find the top three paid employees. (pass3)

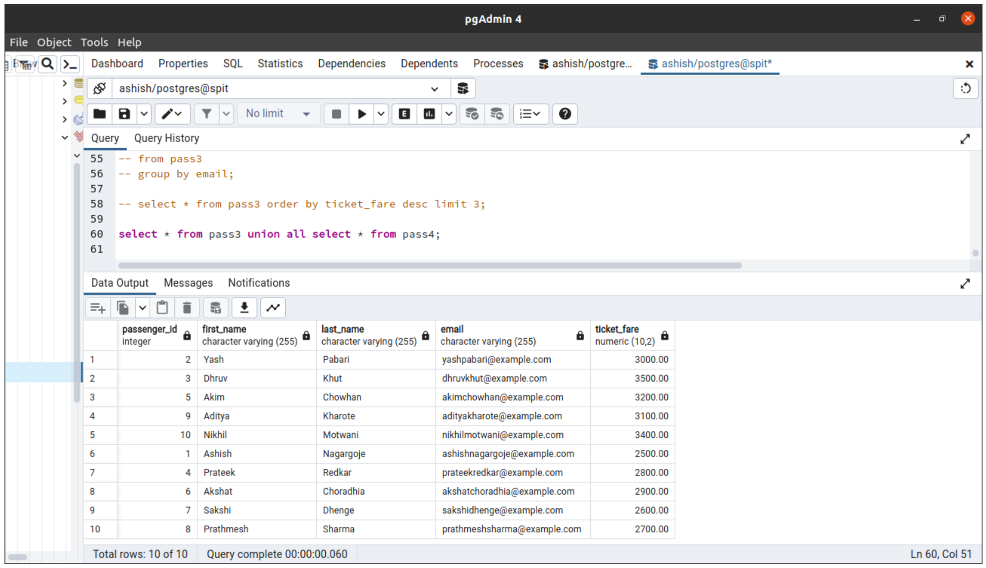


2. For the horizontal fragments check the correctness rules:

Completeness: If relation R is decomposed into fragments R1, R2, ... Rn each data item that can be found in R can also be found in one or more Ri's.

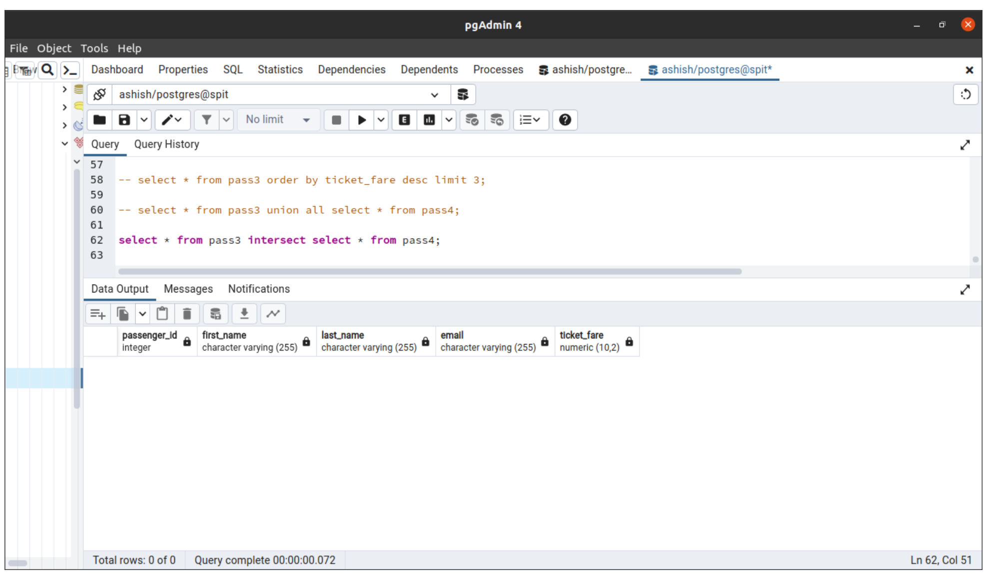


Reconstruction: If relation R is decomposed into fragments R1, R2, ... Rn, it should be possible to define relational operator delta such that R= delta (Ri) for all Ri belongs to Fr.



Thus, after performing UNION ALL on fragmentation it gives original schema.

Disjointness: If relation R is horizontally decomposed into fragments R1, R2... Rn and data item di is in Rj it is not in any other fragment Rk (k not equal to 1)



Thus, we can say that tuples from fragment emp3 does not appear in fragment emp4

Conclusion:

1. In the first set of queries, we fragment the passengers table horizontally using the ticket fare as the parameter.
2. In the second set of queries, we fragmented the pass3 and pass4 tables horizontally using ticket fare as the parameter.
3. We then run different queries to observe the values in the two fragmented tables. The fragmented values are then checked using different queries. We can notice in the above queries that the fragmented tables are disjoint.
4. Furthermore, to prove that these disjoint sets contain all the values of the original table, we join the two sets.
5. In these joins, we observe that we retrieve the original table back. This suggests that even when we broke the table into pieces it remained consistent and had all the entries required.
6. From the above, we can successfully conclude that the relation between the new tables formed and the original table has completeness, reconstruction, and disjointness. That is, it follows all those properties.

Hence, we were able to demonstrate how to design a distributed database by applying the concept of horizontal fragmentation. Additionally, we demonstrated the concept of distributed systems, namely, completeness, reconstruction and disjointness.