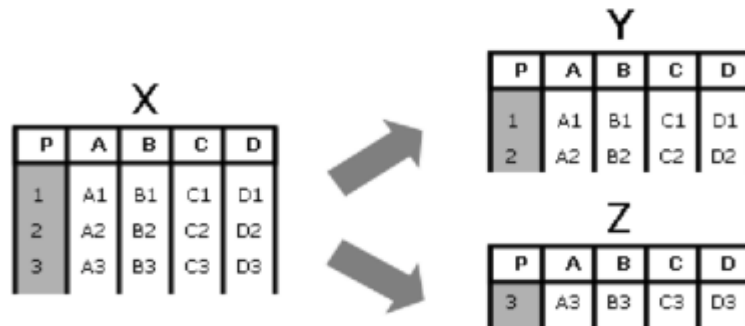


<b>Name:</b>	Shubham Golwal
<b>UID no.</b>	2020300015
<b>Experiment No.</b>	03

<b>AIM:</b>	Horizontal fragmentation
<b>Program 1</b>	
<b>Problem Statement :</b>	Design a distributed database system by applying the concept of horizontal fragmentation.
<b>Theory :</b>	<p><b>What is Fragmentation?</b></p> <p>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.</p> <p>There are two main types of fragmentation in ADBMS:</p> <ol style="list-style-type: none"> <li>1. Horizontal Fragmentation</li> <li>2. Vertical Fragmentation</li> </ol> <p><b>Horizontal Fragmentation</b></p> <p>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.</p>



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  $R = \delta(R_i)$  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.

	<p><b>Security</b> Data not required by local applications is not stored, and consequently not available to unauthorized users.</p> <p><b>Disadvantages of fragmentation</b> Fragmentation has two primary disadvantages, which we have mentioned previously:</p> <p><b>Performance</b> The performance of global application that requires data from several fragments located at different sites may be slower.</p> <p><b>Integrity</b> Integrity control may be more difficult if data and functional dependencies are fragmented and located at different sites.</p>
<b>Queries:</b>	<pre>-- 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</pre>

```
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, first\_name, last\_name, Email, ticket\_fare) and insert 10 records.

The screenshot shows the pgAdmin 4 interface. The left sidebar displays the database structure, including Schemas (1) and Tables (5). The main pane shows a SQL query being executed. The query is as follows:

```
16 -- CREATE TABLE public.passengers
17 -- ALTER TABLE IF EXISTS public.passengers
18 -- OWNER to postgres;
19
20 -- select * from passengers;
21
22 Create table pass3 as
23 select * from passengers
24 where ticket_fare >= 3000;
25
26 select * from pass3;
```

The Data Output pane shows the results of the query, displaying 5 rows of data from the 'pass3' table. The columns are passenger\_id, first\_name, last\_name, email, and ticket\_fare.

passenger_id	first_name	last_name	email	ticket_fare
1	Yash	Pabari	yashpabari@example.com	3000.00
2	Dhruv	Khut	dhruvkhut@example.com	3500.00
3	Akim	Chowhan	akimchowhan@example.com	3200.00
4	Aditya	Kharote	adityakharote@example.com	3100.00
5	Nikhil	Motwani	nikhilmotwani@example.com	3400.00

Total rows: 5 of 5 Query complete 00:00:00.063 Ln 23, Col 26

The screenshot shows the pgAdmin 4 interface. The left sidebar displays the database structure, including Schemas (1) and Tables (5). The main pane shows a SQL query being executed. The query is as follows:

```
18 -- OWNER to postgres;
19
20 select * from passengers;
```

The Data Output pane shows the results of the query, displaying 10 rows of data from the 'passengers' table. The columns are passenger\_id, first\_name, last\_name, email, and ticket\_fare.

passenger_id	first_name	last_name	email	ticket_fare
1	Ashish	Nagargoje	ashishnagargoje@example.com	2500.00
2	Yash	Pabari	yashpabari@example.com	3000.00
3	Dhruv	Khut	dhruvkhut@example.com	3500.00
4	Prateek	Redkar	prateekredkar@example.com	2800.00
5	Akim	Chowhan	akimchowhan@example.com	3200.00
6	Akshat	Choradhia	akshatchoradhia@example.com	2900.00
7	Sakshi	Dhenge	sakshidhenge@example.com	2600.00
8	Prathmesh	Sharma	prathmeshsharma@example.com	2700.00
9	Aditya	Kharote	adityakharote@example.com	3100.00
10	Nikhil	Motwani	nikhilmotwani@example.com	3400.00

Total rows: 10 of 10 Query complete 00:00:00.058 Ln 20, Col 26

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

pass3 contains tuples whose  
salary  $\geq 3000$

The screenshot shows the pgAdmin 4 interface. The left sidebar displays the database structure, including the 'public' schema and its tables. The main pane shows a SQL query editor with the following code:

```
-- 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;
```

The 'Data Output' tab shows the results of the query, displaying a table with 5 rows and 5 columns: passenger\_id, first\_name, last\_name, email, and ticket\_fare.

passenger_id	first_name	last_name	email	ticket_fare
1	Ashish	Nagargoje	ashishnagargoje@example.com	2500.00
2	Prateek	Redkar	prateekredkar@example.com	2800.00
3	Akshat	Choradhia	akshatchoradhia@example.com	2900.00
4	Sakshi	Dhenge	sakshidhenge@example.com	2600.00
5	Prathmesh	Sharma	prathmeshsharma@example.com	2700.00

Total rows: 5 of 5 Query complete 00:00:00.064 Ln 32, Col 20

pass4 contains tuples whose salary  $< 3000$

The screenshot shows the pgAdmin 4 interface. The left sidebar displays the database structure, including the 'public' schema and its tables. The main pane shows a SQL query editor with the following code:

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

-- select * from pass3;

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

-- select * from pass4;

select first_name, last_name from pass3 union all select first_name, las
```

The 'Data Output' tab shows an error message:

```
ERROR: each UNION query must have the same number of columns
LINE 34: ...first_name, last_name from pass3 union all select first_name...
```

SQL state: 42601  
Character: 921

Total rows: 10 of 10 Query complete 00:00:00.052 Ln 34, Col 1

Execute following queries on horizontal fragments:

a. Find ticket price of all Passengers

pgAdmin 4

File Object Tools Help

ashish/postgres@spit

Query

```

27
28 -- Create table pass4 as
29 -- select * from passengers
30 -- where ticket_fare < 3000;
31
32 -- select * from pass4;
33
34 select first_name, last_name, ticket_fare from pass3 union all select first_name, last_name, ticket_fare from pass4;

```

Data Output Messages Notifications

	first_name	last_name	ticket_fare
1	Yash	Pabari	3000.00
2	Dhruv	Khut	3500.00
3	Akim	Chowhan	3200.00
4	Aditya	Kharote	3100.00
5	Nikhil	Motwani	3400.00
6	Ashish	Nagargoje	2500.00
7	Prateek	Redkar	2800.00
8	Akshat	Choradhia	2900.00
9	Sakshi	Dhenge	2600.00
10	Prathmesh	Sharma	2700.00

Total rows: 10 of 10 Query complete 00:00:00.051 Ln 34, Col 42

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

pgAdmin 4

File Object Tools Help

ashish/postgres@spit

Query

```

25
26 -- select * from pass3;
27
28 -- Create table pass4 as
29 -- select * from passengers
30 -- where ticket_fare < 3000;
31
32 -- select * from pass4;
33
34 -- union
35 -- select first_name, last_name, ticket_fare from pass3 union all select first_name, last_name, ticket_fare from pass4;
36
37 -- find
38 select email from pass3 where ticket_fare = 2500;

```

Data Output Messages Notifications

	email
--	-------

Total rows: 0 of 0 Query complete 00:00:00.060 Ln 38, Col 50

c. Find first\_name, last\_name and ticket price where passenger id is known

pgAdmin 4

File Object Tools Help

Dashboard Properties SQL Statistics Dependencies Dependents Processes ashish/postgre... ashish/postgres@spit\*

ashish/postgres@spit

Query Query History

```

35 -- select first_name, last_name, ticket_fare from pass3 union all select first_name, last_name, ticket_fare from pass4;
36
37 -- find
38 -- select email from pass3 where ticket_fare = 2500;
39
40 select first_name, last_name, ticket_fare from pass3 where passenger_id = 1 union all select first_name, last_name,
41 ticket_fare from pass4 where passenger_id = 1;
42

```

Data Output Messages Notifications

first_name	last_name	ticket_fare
character varying (255)	character varying (255)	numeric (10,2)
1 Ashish	Nagargoje	2500.00

Total rows: 1 of 1 Query complete 00:00:00.053 Ln 41, Col 1

d. Find the first\_name, last\_name, email and ticket price where passenger id is known

pgAdmin 4

File Object Tools Help

Dashboard Properties SQL Statistics Dependencies Dependents Processes ashish/postgre... ashish/postgres@spit\*

ashish/postgres@spit

Query Query History

```

37
38 mail from pass3 where ticket_fare = 2500;
39
40 t_name, last_name, ticket_fare from pass3 where passenger_id = 1 union all select first_name, last_name, ticket_fare from pass4 where
41
42 t_name, last_name, ticket_fare, email from pass3 where passenger_id = 1 union all select first_name, last_name, ticket_fare, email| fro
43

```

Data Output Messages Notifications

first_name	last_name	ticket_fare	email
character varying (255)	character varying (255)	numeric (10,2)	character varying (255)
1 Ashish	Nagargoje	2500.00	ashishnagargoje@example.com

Total rows: 1 of 1 Query complete 00:00:00.068 Ln 42, Col 142

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



pgAdmin 4

File Object Tools Help

Dashboard Properties SQL Statistics Dependencies Dependents Processes ashish/postgre... ashish/postgres@spit\*

ashish/postgres@spit

Query Query History

```

40 -- select first_name, last_name, ticket_fare from pass3 where passenger_id = 1 union all select first_name, last_name, ticket_fare fro
41
42 -- select first_name, last_name, ticket_fare, email from pass3 where passenger_id = 1 union all select first_name, last_name, ticket_f
43
44 select sum(ticket_fare)
45 from pass4
46 where ticket_fare < 2900;

```

Data Output Messages Notifications

sum
numeric
1 10600.00

Total rows: 1 of 1 Query complete 00:00:00.032 Ln 46, Col 21

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

pgAdmin 4

File Object Tools Help

Dashboard Properties SQL Statistics Dependencies Dependents Processes ashish/postgre... ashish/postgres@spit\*

ashish/postgres@spit

Query Query History

```

46 -- from pass4
47 -- where ticket_fare < 2900;
48
49 -- sum -> null
50 select sum(ticket_fare)
51 from pass4
52 where ticket_fare < 1000;

```

Data Output Messages Notifications

sum
numeric
1 [null]

Successfully run. Total query runtime: 79 msec. 1 rows affected. X

Total rows: 1 of 1 Query complete 00:00:00.079 Ln 52, Col 23

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

pgAdmin 4

FileObjectToolsHelp

ashish/postgres@spit

DashboardPropertiesSQLStatisticsDependenciesDependentsProcessesashish/postgre...ashish/postgres@spit\*

ashish/postgres@spit

QueryHistory

53

-- select count(passenger\_id), email

-- from pass3

-- group by email;

57

select \* from pass3 order by ticket\_fare desc limit 3;

59

Data OutputMessagesNotifications

passenger_id integer	first_name character varying (255)	last_name character varying (255)	email character varying (255)	ticket_fare numeric (10,2)
1	3Dhruv	Khut	dhruvkhut@example.com	3500.00
2	10Nikhil	Motwani	nikhilmotwani@example.com	3400.00
3	5Akim	Chowhan	akimchowhan@example.com	3200.00

Total rows: 3 of 3

Query complete 00:00:00.041

Ln 58, Col 42

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.

pgAdmin 4

FileObjectToolsHelp

DashboardPropertiesSQLStatisticsDependenciesDependentsProcessesashish/postgre...ashish/postgres@spit\*

ashish/postgres@spit

No limit

QueryQuery History

55-- from pass3

56-- group by email;

57

58-- select \* from pass3 order by ticket\_fare desc limit 3;

59

60select \* from pass3 union all select \* from pass4;

61

Data OutputMessagesNotifications

passenger_id integer	first_name character varying (255)	last_name character varying (255)	email character varying (255)	ticket_fare numeric (10,2)
1	2Yash	Pabari	yashpabari@example.com	3000.00
2	3Dhruv	Khut	dhruvkhut@example.com	3500.00
3	5Akim	Chowhan	akimchowhan@example.com	3200.00
4	9Aditya	Kharote	adityakharote@example.com	3100.00
5	10Nikhil	Motwani	nikhilmotwani@example.com	3400.00
6	1Ashish	Nagargoje	ashishnagargoje@example.com	2500.00
7	4Prateek	Redkar	prateekredkar@example.com	2800.00
8	6Akshat	Choradhia	akshatchoradhia@example.com	2900.00
9	7Sakshi	Dhenge	sakshidhenge@example.com	2600.00
10	8Prathmesh	Sharma	prathmeshshama@example.com	2700.00

Total rows: 10 of 10Query complete 00:00:00.060Ln 60, Col 51

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

The screenshot shows a pgAdmin 4 window with a query editor containing the following SQL code:

```

55 -- from pass3
56 -- group by email;
57
58 -- select * from pass3 order by ticket_fare desc limit 3;
59
60 select * from pass3 union all select * from pass4;
61

```

The Data Output tab displays the result of the query as a table with 10 rows and 6 columns:

passenger_id	first_name	last_name	email	ticket_fare
1	Yash	Pabari	yashpabari@example.com	3000.00
2	Dhruv	Khut	dhruvkhut@example.com	3500.00
3	Akim	Chowhan	akimchowhan@example.com	3200.00
4	Aditya	Kharote	adityakharote@example.com	3100.00
5	Nikhil	Motwani	nikhimotwani@example.com	3400.00
6	Ashish	Nagargoje	ashishnagargoje@example.com	2500.00
7	Prateek	Redkar	prateekredkar@example.com	2800.00
8	Akshat	Choradhia	akshatchoradhia@example.com	2900.00
9	Sakshi	Dhenge	sakshidhenge@example.com	2600.00
10	Prathmesh	Sharma	prathmeshsharma@example.com	2700.00

Total rows: 10 of 10 Query complete 00:00:00.060 Ln 60, Col 51

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)

The screenshot shows a pgAdmin 4 window with a query editor containing the following SQL code:

```

57
58 -- select * from pass3 order by ticket_fare desc limit 3;
59
60 -- select * from pass3 union all select * from pass4;
61
62 select * from pass3 intersect select * from pass4;
63

```

The Data Output tab displays the result of the query as a table with 5 columns:

passenger_id	first_name	last_name	email	ticket_fare
--------------	------------	-----------	-------	-------------

Total rows: 0 of 0 Query complete 00:00:00.072 Ln 62, Col 51

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.