

Experiment 3: Vertical Fragmentation

- Name: Srikanth Iyengar
- UID: 2020400062
- Branch: IT
- Batch: G
- Subject: Advanced Database Management System

Aim:

Design a distributed database by applying the concept of vertical fragmentation

Implementation

Query for creation of table

```
drop table if exists bank_details;

create table bank_details (
    acc_no integer ,
    cust_name varchar(100),
    mobile_no bigint,
    branch varchar(100),
    balance integer,
    loan_amount integer,
    amount_due integer,
    customer_id integer,
    date_of_birth date,
    transaction_no varchar(100),
    date_of_transaction date,
    mode_of_transaction varchar(100),
    transaction_type varchar(100),
    transaction_amount integer,
    primary key (acc_no, customer_id)
);

drop table if exists customer, customer_transaction, customer_loan;
INSERT INTO bank_details (acc_no, cust_name, mobile_no, branch, balance,
loan_amount, amount_due, customer_id, date_of_birth, transaction_no,
date_of_transaction, mode_of_transaction, transaction_type, transaction_amount)
VALUES
    (1001, 'John Doe', 9876543210, 'Central Branch', 5000, 10000, 500, 101, '1980-01-01', 'T1001', '2023-02-01', 'Cash', 'Deposit', 1000),
    (1002, 'Jane Doe', 9876543211, 'Central Branch', 6000, 20000, 1000, 102, '1982-02-01', 'T1002', '2023-02-02', 'Cheque', 'Withdrawal', 500),
    (1003, 'John Smith', 9876543212, 'West Branch', 7000, 30000, 1500, 103, '1984-03-01', 'T1003', '2023-02-03', 'Net Banking', 'Deposit', 2000),
    (1004, 'Jane Smith', 9876543213, 'West Branch', 8000, 40000, 2000, 104, '1986-04-
```

```

01', 'T1004', '2023-02-04', 'Cash', 'Withdrawal', 800),
(1005, 'Bob Brown', 9876543214, 'South Branch', 9000, 50000, 2500, 105, '1988-05-01', 'T1005', '2023-02-05', 'Cheque', 'Deposit', 1500),
(1006, 'Alice Brown', 9876543215, 'South Branch', 10000, 60000, 3000, 106, '1990-06-01', 'T1006', '2023-02-06', 'Net Banking', 'Withdrawal', 700),
(1007, 'Charlie Smith', 9876543216, 'East Branch', 11000, 70000, 3500, 107, '1992-07-01', 'T1007', '2023-02-07', 'Cash', 'Deposit', 1700),
(1008, 'Emily Smith', 9876543217, 'East Branch', 12000, 80000, 4000, 108, '1994-08-01', 'T1008', '2023-02-08', 'Cheque', 'Withdrawal', 900),
(1009, 'David Brown', 9876543218, 'North Branch', 13000, 90000, 4500, 109, '1996-09-01', 'T1009', '2023-02-09', 'Net Banking', 'Deposit', 1900),
(1010, 'Emma Brown', 9876543219, 'North Branch', 14000, 100000, 5000, 110, '1998-10-01', 'T1010', '2023-02-10', 'Cash', 'Withdrawal', 1100);
select * from bank_details;

```

```

create table customer (
    acc_no integer ,
    cust_name varchar(100),
    mobile_no bigint,
    branch varchar(100),
    balance integer,
    date_of_birth date,
    customer_id integer,
    primary key (acc_no, customer_id)
);

```

```

insert into customer (select acc_no, cust_name, mobile_no, branch, balance,
date_of_birth, customer_id from bank_details);
select * from customer;

```

```

create table customer_loan (
    acc_no integer,
    amount_due integer,
    customer_id integer,
    loan_amount integer,
    primary key (acc_no, customer_id)
);

```

```

insert into customer_loan (select acc_no, amount_due, customer_id, loan_amount from
bank_details);

```

```

drop table if exists customer_transaction;
create table customer_transaction (
    acc_no integer,
    customer_id integer,
    transaction_no varchar(100),
    date_of_transaction date,
    transaction_type varchar(100),
    mode_of_transaction varchar(100),
    transaction_amount integer,
    primary key (acc_no, customer_id)
);
insert into customer_transaction (select acc_no, customer_id, transaction_no,
date_of_transaction, transaction_type , mode_of_transaction , transaction_amount
from bank_details);
select * from (customer join customer_loan using (acc_no) ) join
customer_transaction using (acc_no);
select * from customer_transaction;

```

PART A

Query 1

```

-- find all the customer with branch ANDHERI
SELECT * FROM (
    SELECT * FROM (
        (SELECT * FROM customer WHERE branch = 'ANDHERI') y join (SELECT * FROM
customer_loan) z using (acc_no)
    ) x JOIN (SELECT * FROM customer_transaction) q USING (acc_no)) w;

```

Result From table 1

	acc_no integer	cust_name character varying (100)	mobile_no bigint	branch character varying (100)	balance integer	date_of_birth date	customer_id integer	amount_due integer	customer_id integer	loan_amount integer	customer_id integer
1	1003	John Smith	9876543212	West Branch	7000	1984-03-01	103	1500	103	30000	103
2	1004	Jane Smith	9876543213	West Branch	8000	1986-04-01	104	2000	104	40000	104

Query 2

```

-- find all the customer who have done NET BANKING transaction
SELECT * FROM (
    SELECT * FROM (
        (SELECT * FROM customer) y join (SELECT * FROM customer_loan) z using (acc_no)
    ) x JOIN (SELECT * FROM customer_transaction WHERE transaction_type = 'Deposit') q
    USING (acc_no)) w;

```

Result From table 1

Servers (7)

- Another Supabase
- CockroachDB Postgres
- NEON DB
- Postgres supabase
 - Databases (4)
 - expt1
 - expt2
 - expt3
 - Casts
 - Catalogs
 - Event Triggers
 - Extensions
 - Foreign Data Wrappers
 - Languages
 - Publications
 - Schemas
 - Subscriptions
 - postgres
 - Login/Group Roles
 - Tablespaces
- Railway.app Postgres
- Render DB
- bitlio

exp3/postgres@Postgres supabase

Query

Query History

```
1 -- find all the customer with branch ANDHERI
2 SELECT * FROM bank_details where branch = 'ANDHERI';
3 SELECT * FROM (
4   SELECT * FROM (
5     (SELECT * FROM customer WHERE branch = 'West Branch') y join (SELECT * FROM customer_loan) z using (acc_no)
6   ) x JOIN (SELECT * FROM customer_transaction) q USING (acc_no)) w;
7 -- find all the customer who have done NET BANKING transaction
8 SELECT * FROM (
9   SELECT * FROM (
10    (SELECT * FROM customer) y join (SELECT * FROM customer_loan) z using (acc_no)
11  ) x JOIN (SELECT * FROM customer_transaction WHERE transaction_type = 'Deposit') q USING (acc_no)) w;
12 -- find all the customer of transaction type NET BANKING
```

Data Output

Messages

Notifications

	acc_no integer	cust_name character varying (100)	mobile_no bigint	branch character varying (100)	balance integer	date_of_birth date	customer_id integer	amount_due integer	customer_id integer	loan_amount integer	customer_id integer
1	1001	John Doe	9876543210	Central Branch	5000	1980-01-01	101	500	101	10000	101
2	1003	John Smith	9876543212	West Branch	7000	1984-03-01	103	1500	103	30000	103
3	1005	Bob Brown	9876543214	South Branch	9000	1988-05-01	105	2500	105	50000	105
4	1007	Charlie Smith	9876543216	East Branch	11000	1992-07-01	107	3500	107	70000	107
5	1009	David Brown	9876543218	North Branch	13000	1996-09-01	109	4500	109	90000	109

Successfully run. Total query runtime: 1 secs 169 msec. 5 rows affected.

Total rows: 5 of 5 Query complete 00:00:01.169 Ln 8, Col 1

Query 3

```
-- find all the customer of transaction type NET BANKING
SELECT * FROM (
  SELECT * FROM (
    (SELECT * FROM customer) y join (SELECT * FROM customer_loan) z using (acc_no)
  ) x JOIN (SELECT * FROM customer_transaction WHERE mode_of_transaction = 'Net
Banking') q USING (acc_no)) w;
```

Result DB from table 1

Servers (7)

- Another Supabase
- CockroachDB Postgres
- NEON DB
- Postgres supabase
 - Databases (4)
 - expt1
 - expt2
 - expt3
 - Casts
 - Catalogs
 - Event Triggers
 - Extensions
 - Foreign Data Wrappers
 - Languages
 - Publications
 - Schemas
 - Subscriptions
 - postgres
 - Login/Group Roles
 - Tablespaces
- Railway.app Postgres
- Render DB
- bitlio

exp3/postgres@Postgres supabase

Query

Query History

```
8 SELECT * FROM (
9   SELECT * FROM (
10    (SELECT * FROM customer) y join (SELECT * FROM customer_loan) z using (acc_no)
11  ) x JOIN (SELECT * FROM customer_transaction WHERE transaction_type = 'Deposit') q USING (acc_no)) w;
12 -- find all the customer of transaction type NET BANKING
13 SELECT * FROM (
14   SELECT * FROM (
15    (SELECT * FROM customer) y join (SELECT * FROM customer_loan) z using (acc_no)
16  ) x JOIN (SELECT * FROM customer_transaction WHERE mode_of_transaction = 'Net Banking') q USING (acc_no)) w;
17 -- find all the customer who are of atleast 20 years of age
18 SELECT * FROM (
19   SELECT * FROM (
```

Data Output

Messages

Notifications

	acc_no integer	cust_name character varying (100)	mobile_no bigint	branch character varying (100)	balance integer	date_of_birth date	customer_id integer	amount_due integer	customer_id integer	loan_amount integer	customer_id integer
1	1003	John Smith	9876543212	West Branch	7000	1984-03-01	103	1500	103	30000	103
2	1006	Alice Brown	9876543215	South Branch	10000	1990-06-01	106	3000	106	60000	106
3	1009	David Brown	9876543218	North Branch	13000	1996-09-01	109	4500	109	90000	109

Successfully run. Total query runtime: 1 secs 108 msec. 3 rows affected.

Total rows: 3 of 3 Query complete 00:00:01.108 Ln 12, Col 1

Query 4

```
-- find all the customer who are of atleast 20 years of age
SELECT * FROM (
    SELECT * FROM (
        (SELECT * FROM customer where date_part('year', current_date) -
date_part('year', date_of_birth) > 35 ) y join (SELECT * FROM customer_loan) z using
(acc_no)
    ) x JOIN (SELECT * FROM customer_transaction ) q USING (acc_no)) w;
```

Result From table 1

The screenshot shows the pgAdmin 4 interface. The left sidebar displays a tree view of the database structure, including servers, databases, and tables. The main pane shows a SQL query being executed. The query is a complex join involving customer, customer_loan, and customer_transaction tables, filtering for customers with a balance of at least 3500 and a transaction mode of 'Net Banking'.

Query:

```
-- find all the customer of transaction type NET BANKING
SELECT * FROM (
    SELECT * FROM (
        (SELECT * FROM customer) y join (SELECT * FROM customer_loan) z using (acc_no)
    ) x JOIN (SELECT * FROM customer_transaction WHERE mode_of_transaction = 'Net Banking') q USING (acc_no)) w;
-- find all the customer who are of atleast 20 years of age
SELECT * FROM (
    SELECT * FROM (
        (SELECT * FROM customer where date_part('year', current_date) - date_part('year', date_of_birth) > 35 ) y join (SELECT * FROM customer_transaction) z using (acc_no)
    ) x JOIN (SELECT * FROM customer_transaction ) q USING (acc_no)) w;
-- find all the customer who less than 3000 amount of due
SELECT * FROM (
    SELECT * FROM (
        (SELECT * FROM customer where date_part('year', current_date) - date_part('year', date_of_birth) > 35 ) y join (SELECT * FROM customer_loan) z using (acc_no)
    ) x JOIN (SELECT * FROM customer_transaction ) q USING (acc_no)) w;
```

Data Output:

acc_no	cust_name	mobile_no	branch	balance	date_of_birth	customer_id	amount_due	customer_id	loan_amount	customer_id
1001	John Doe	9876543210	Central Branch	5000	1980-01-01	101	500	101	10000	101
1002	Jane Doe	9876543211	Central Branch	6000	1982-02-01	102	1000	102	20000	102
1003	John Smith	9876543212	West Branch	7000	1984-03-01	103	1500	103	30000	103
1004	Jane Smith	9876543213	West Branch	8000	1986-04-01	104	2000	104	40000	104

Successfully run. Total query runtime: 1 secs 129 msec. 4 rows affected.

Query 5

```
-- Find all the customer who less than 3000 amount of due
SELECT * FROM (
    SELECT * FROM (
        (SELECT * FROM customer ) y join (SELECT * FROM customer_loan where amount_due <
3000) z using (acc_no)
    ) x JOIN (SELECT * FROM customer_transaction ) q USING (acc_no)) w;
```

Result From table 1

The screenshot shows the pgAdmin interface with a SQL query executed. The query is a complex join involving customer, customer_transaction, and customer_loan tables, filtering for customers aged 35 and above with a loan amount due less than 3000. The result table displays 5 rows of data.

acc_no	cust_name	mobile_no	branch	balance	date_of_birth	customer_id	amount_due	customer_id	loan_amount	customer_id
1001	John Doe	9876543210	Central Branch	5000	1980-01-01	101	500	101	10000	101
1002	Jane Doe	9876543211	Central Branch	6000	1982-02-01	102	1000	102	20000	102
1003	John Smith	9876543212	West Branch	7000	1984-03-01	103	1500	103	30000	103
1004	Jane Smith	9876543213	West Branch	8000	1986-04-01	104	2000	104	40000	104
1005	Bob Brown	9876543214	South Branch	9000	1988-05-01	105	2500	105	50000	105

Successfully run. Total query runtime: 1 secs 81 msec. 5 rows affected.

Query 6

```
-- Find all the customer who has a account in branch
SELECT sum(loan_amount) FROM (
    SELECT * FROM (
        (SELECT * FROM customer WHERE branch = 'Central Branch') y join (SELECT * FROM
customer_loan) z using (acc_no)
    ) x JOIN (SELECT * FROM customer_transaction ) q USING (acc_no)) w;
```

Result From table 1

The screenshot shows the pgAdmin interface with a simplified SQL query executed. The query calculates the sum of loan amounts for customers in the 'Central Branch' who have a loan. The result table shows a single row with a sum of 30000.

sum
30000

Successfully run. Total query runtime: 1 secs 94 msec. 1 rows affected.

PART B

Implementation of part B

Flight Table

Flight Id	Flight Name	Source	Destination	Tickets Available
1	Transatlantic Express	New York	London	100
2	City Hopper	London	Paris	90
3	European Voyager	Paris	Berlin	80
4	Continent Cruiser	Berlin	Rome	70
5	Mediterranean Flyer	Rome	Madrid	60
6	Iberian Explorer	Madrid	Barcelona	50
7	Nordic Adventurer	Barcelona	Amsterdam	40
8	Oceanic Odyssey	Amsterdam	New York	30
9	Atlantic Ascension	New York	London	20
10	Skyline Shuttle	London	Paris	10
11	Continental Commuter	Paris	Berlin	20
12	Bella Italia	Berlin	Rome	30
13	España Express	Rome	Madrid	40
14	Dutch Delight	Madrid	Barcelona	50
15	Viking Venture	Barcelona	Amsterdam	60
16	Big Apple Bounce	Amsterdam	New York	70
17	British Airways	New York	London	80
18	French Connection	London	Paris	90
19	German Getaway	Paris	Berlin	100
20	Italian Holiday	Berlin	Rome	110

App user Table

Username	Password	Tickets Booked	Location
johndoe123	5f4dcc3b5aa765d61d8327deb882cf99	2	New York
janedoe456	5e884898da28047151d0e56f8dc62927	1	London
bobsmith789	21232f297a57a5a743894a0e4a801fc3	4	Paris
alicebrown101	ee11cbb19052e40b07aac0ca060c23ee	0	Berlin

tomgreen202	25f9e794323b453885f5181f1b624d0b	3	Madrid
jennypink303	a665a45920422f9d417e4867efdc4fb8	5	Rome
chrisblack404	5d41402abc4b2a76b9719d911017c592	6	Moscow
kimwhite505	bdbafc1b5f5dc8ab2f0196b5cd543a33	7	Toronto
paulred606	1a1dc91c907325c69271ddf0c944bc72	8	Sydney
lucyyellow707	ac0eafeed417f0ffa7befa0d1c01a7a6	9	Seoul
mikeblue808	098f6bcd4621d373cade4e832627b4f6	10	Tokyo
sarahpurple909	0b7c75b0f4b7e585a1c73e72409f3b7c	11	Shanghai
davidorange010	76a2173be6393254e72ffa4d6df1030a	12	Beijing
emmablack112	c1dfd96eea8cc2b62785275bca38ac26	13	Mexico City
danielgreen213	98f13708210194c475687be6106a3bce	14	Rio de Janeiro
elizabethpurple314	f0d2f1e2dc9abceb877557a483e07c3a	15	Sao Paulo
richardblue415	5ab5dbc27120680fa6f0b6ed7b6c0098	16	Buenos Aires
jenniferred516	7c222fb2927d828af22f592134e89324	17	Lima
robertyellow617	7a74f16c3e7d20a03b356f7bcf8c8f7a	18	Bogota

Query for creation of required tables

```
drop table if exists app_user;

CREATE TABLE IF NOT EXISTS public.app_user
(
    username character varying(100) COLLATE pg_catalog."default" NOT NULL,
    user_password character varying(100) COLLATE pg_catalog."default",
    tickets_booked integer,
    address varchar(100),
    CONSTRAINT app_user_pkey PRIMARY KEY (username)
);

INSERT INTO public.app_user (username, user_password, tickets_booked, address)
VALUES
('johndoe123', '5f4dcc3b5aa765d61d8327deb882cf99', 2, 'New York'),
('janedoe456', '5e884898da28047151d0e56f8dc62927', 1, 'London'),
('bobsmith789', '21232f297a57a5a743894a0e4a801fc3', 4, 'Paris'),
('alicebrown101', 'ee11cbb19052e40b07aac0ca060c23ee', 0, 'Berlin'),
('tomgreen202', '25f9e794323b453885f5181f1b624d0b', 3, 'Madrid'),
('jennypink303', 'a665a45920422f9d417e4867efdc4fb8', 5, 'Rome'),
('chrisblack404', '5d41402abc4b2a76b9719d911017c592', 6, 'Moscow'),
('kimwhite505', 'bdbafc1b5f5dc8ab2f0196b5cd543a33', 7, 'Toronto'),
('paulred606', '1a1dc91c907325c69271ddf0c944bc72', 8, 'Sydney'),
```



```
(
    'lucyyellow707', 'ac0eafeed417f0ffa7befa0d1c01a7a6', 9, 'Seoul'),
    ('mikeblue808', '098f6bcd4621d373cade4e832627b4f6', 10, 'Tokyo'),
    ('sarahpurple909', '0b7c75b0f4b7e585a1c73e72409f3b7c', 11, 'Shanghai'),
    ('davidorange010', '76a2173be6393254e72ffa4d6df1030a', 12, 'Beijing'),
    ('emmablack112', 'c1dfd96eea8cc2b62785275bca38ac26', 13, 'Mexico City'),
    ('danielgreen213', '98f13708210194c475687be6106a3bce', 14, 'Rio de Janeiro'),
    ('elizabethpurple314', 'f0d2f1e2dc9abceb877557a483e07c3a', 15, 'Sao Paulo'),
    ('richardblue415', '5ab5dbc27120680fa6f0b6ed7b6c0098', 16, 'Buenos Aires'),
    ('jenniferred516', '7c222fb2927d828af22f592134e89324', 17, 'Lima'),
    ('robertyellow617', '7a74f16c3e7d20a03b356f7bcf8c8f7a', 18, 'Bogota');

```

```
drop table if exists flight;
```

```
CREATE TABLE IF NOT EXISTS public.flight
(
    flight_id integer NOT NULL,
    flight_name character varying(100) COLLATE pg_catalog."default" NOT NULL,
    flight_source character varying(100) COLLATE pg_catalog."default" NOT NULL,
    flight_destination character varying(100) COLLATE pg_catalog."default" NOT NULL,
    tickets_available integer NOT NULL,
    CONSTRAINT flight_pkey PRIMARY KEY (flight_id)
);

```

```
insert into flight values
(1, 'Transatlantic Express', 'New York', 'London', 100),
(2, 'City Hopper', 'London', 'Paris', 90),
(3, 'European Voyager', 'Paris', 'Berlin', 80),
(4, 'Continent Cruiser', 'Berlin', 'Rome', 70),
(5, 'Mediterranean Flyer', 'Rome', 'Madrid', 60),
(6, 'Iberian Explorer', 'Madrid', 'Barcelona', 50),
(7, 'Nordic Adventurer', 'Barcelona', 'Amsterdam', 40),
(8, 'Oceanic Odyssey', 'Amsterdam', 'New York', 30),
(9, 'Atlantic Ascension', 'New York', 'London', 20),
(10, 'Skyline Shuttle', 'London', 'Paris', 10),
(11, 'Continental Commuter', 'Paris', 'Berlin', 20),
(12, 'Bella Italia', 'Berlin', 'Rome', 30),
(13, 'España Express', 'Rome', 'Madrid', 40),
(14, 'Dutch Delight', 'Madrid', 'Barcelona', 50),
(15, 'Viking Venture', 'Barcelona', 'Amsterdam', 60),
(16, 'Big Apple Bounce', 'Amsterdam', 'New York', 70),
(17, 'British Airways', 'New York', 'London', 80),
(18, 'French Connection', 'London', 'Paris', 90),
(19, 'German Getaway', 'Paris', 'Berlin', 100),
(20, 'Italian Holiday', 'Berlin', 'Rome', 110);

```

```
DROP TABLE IF EXISTS flight_transaction;
```

```
CREATE TABLE flight_transaction (

```

```

transaction_id SERIAL PRIMARY KEY,
username VARCHAR(100) NOT NULL,
flight_id INTEGER NOT NULL,
tickets_booked INTEGER NOT NULL,
transaction_time TIMESTAMP DEFAULT NOW(),
FOREIGN KEY (username) REFERENCES app_user(username),
FOREIGN KEY (flight_id) REFERENCES flight(flight_id)
);

INSERT INTO flight_transaction (username, flight_id, tickets_booked)
SELECT
    u.username,
    f.flight_id,
    (CASE WHEN u.tickets_booked < f.tickets_available THEN u.tickets_booked ELSE
f.tickets_available END) as tickets_booked
FROM app_user u
INNER JOIN flight f ON (u.address = f.flight_source)
WHERE u.tickets_booked > 0 AND f.tickets_available > 0;

DROP TABLE IF EXISTS big_table;

CREATE TABLE big_table AS
SELECT au.username, au.user_password, au.address, f.flight_id, f.flight_name,
f.flight_source, f.flight_destination, f.tickets_available, ft.transaction_id,
ft.tickets_booked, ft.transaction_time
FROM app_user AS au
JOIN flight_transaction AS ft ON au.username = ft.username
JOIN flight AS f ON ft.flight_id = f.flight_id;

```

Table creation output:

The screenshot shows the pgAdmin 4 web interface. The left sidebar displays a tree view of the database structure, including servers, databases, and schemas. The main panel shows a SQL query being executed in the 'Query' tab. The query history and output are visible below the query editor.

Query:

```

1 drop table if exists flight_transaction, app_user, flight, big_table;
2
3 CREATE TABLE IF NOT EXISTS public.app_user
4 (
5     username character varying(100) COLLATE pg_catalog."default" NOT NULL,
6     user_password character varying(100) COLLATE pg_catalog."default",
7     tickets_booked integer,
8     address varchar(100),
9     CONSTRAINT app_user_pkey PRIMARY KEY (username)
10 );
11
12 INSERT INTO public.app_user (username, user_password, tickets_booked, address)

```

Data Output:

```

NOTICE: table "flight" does not exist, skipping
NOTICE: table "flight_transaction" does not exist, skipping
NOTICE: table "big_table" does not exist, skipping
SELECT 13

```

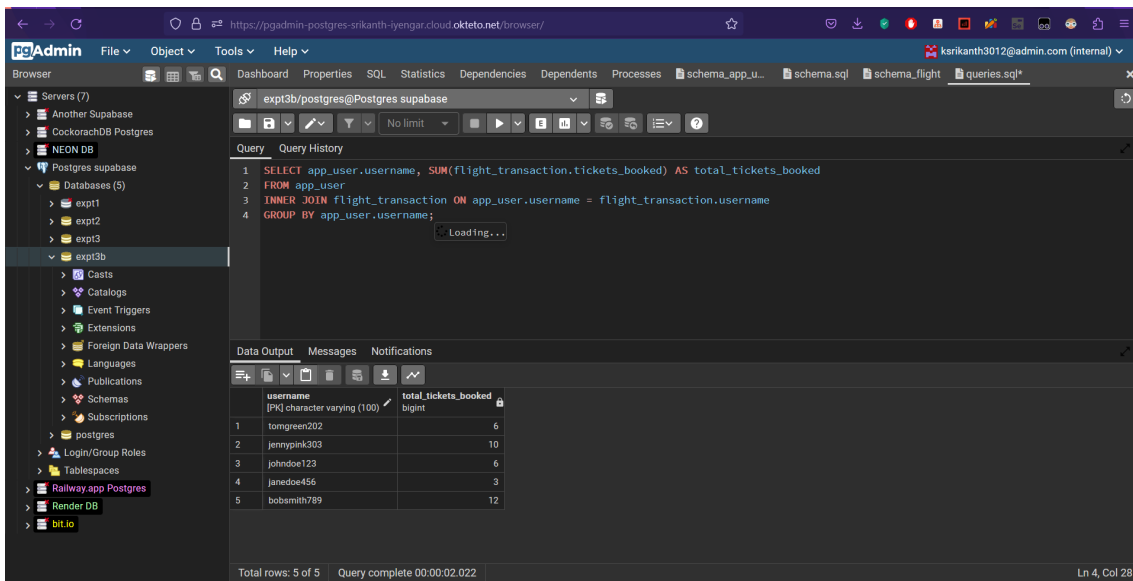
Query returned successfully in 821 msec.

Total rows: 13 of 13 | Query complete 00:00:00.821

Query 1

```
-- What are the total number of tickets booked by each user?
SELECT app_user.username, SUM(flight_transaction.tickets_booked) AS
total_tickets_booked
FROM app_user
INNER JOIN flight_transaction ON app_user.username = flight_transaction.username
GROUP BY app_user.username;
```

Result:



The screenshot shows the pgAdmin interface with the following details:

- Query:** `SELECT app_user.username, SUM(flight_transaction.tickets_booked) AS total_tickets_booked FROM app_user INNER JOIN flight_transaction ON app_user.username = flight_transaction.username GROUP BY app_user.username;`
- Data Output:** A table with 2 columns: `username` (PK character varying (100)) and `total_tickets_booked` (bigint). The results are as follows:

username	total_tickets_booked
tomgreen202	6
jennypink303	10
johnndoe123	6
janedoe456	3
bobsmith789	12

Total rows: 5 of 5 | Query complete 00:00:02.022 | Ln 4, Col 28

Query 2

```
-- What are the flight details of all transactions where more than 10 tickets were
booked?
SELECT app_user.username, SUM(flight_transaction.tickets_booked) AS
total_tickets_bookedSELECT flight_transaction.*, flight.*
FROM flight_transaction
INNER JOIN flight ON flight_transaction.flight_id = flight.flight_id
WHERE flight_transaction.tickets_booked > 3;
```

Result:

pgAdmin interface showing a query execution result. The query is:

```
1 SELECT app_user.username, SUM(flight_transaction.tickets_booked) AS total_tickets_booked
2 FROM app_user
3 INNER JOIN flight_transaction ON app_user.username = flight_transaction.username
4 GROUP BY app_user.username;
5
6 SELECT flight_transaction.*, flight.*
7 FROM flight_transaction
8 INNER JOIN flight ON flight_transaction.flight_id = flight.flight_id
9 WHERE flight_transaction.tickets_booked > 3;
```

The Data Output tab shows the following results:

transaction_id	username	flight_id	tickets_booked	transaction_time	flight_id	flight_name	flight_source	flight_destination
1	7 bobsmith789	19	4	2023-02-16 13:54:28.327385	19	German Getaway	Paris	Berlin
2	8 bobsmith789	11	4	2023-02-16 13:54:28.327385	11	Continental Commuter	Paris	Berlin
3	9 bobsmith789	3	4	2023-02-16 13:54:28.327385	3	European Voyager	Paris	Berlin
4	12 jennypink303	13	5	2023-02-16 13:54:28.327385	13	España Express	Rome	Madrid
5	13 jennypink303	5	5	2023-02-16 13:54:28.327385	5	Mediterranean Flyer	Rome	Madrid

Successfully run. Total query runtime: 1 secs 119 msec. 5 rows affected.

Query 3

```
--What are the total number of tickets booked for each flight, and how many tickets
are available for each flight?

SELECT flight.flight_name, flight.tickets_available,
SUM(flight_transaction.tickets_booked) AS total_tickets_booked
FROM flight
LEFT JOIN flight_transaction ON flight.flight_id = flight_transaction.flight_id
GROUP BY flight.flight_name, flight.tickets_available;
```

Result:

pgAdmin interface showing a query execution result. The query is:

```
5
6 SELECT flight_transaction.*, flight.*
7 FROM flight_transaction
8 INNER JOIN flight ON flight_transaction.flight_id = flight.flight_id
9 WHERE flight_transaction.tickets_booked > 3;
10
11
12 SELECT flight.flight_name, flight.tickets_available, SUM(flight_transaction.tickets_booked) AS total_tickets_booked
13 FROM flight
14 LEFT JOIN flight_transaction ON flight.flight_id = flight_transaction.flight_id
15 GROUP BY flight.flight_name, flight.tickets_available;
16
```

The Data Output tab shows the following results:

flight_name	tickets_available	total_tickets_booked
1 Iberian Explorer	50	3
2 City Hopper	90	1
3 Big Apple Bounce	70	[null]
4 Continental Commuter	20	4
5 Transatlantic Express	100	2
6 Atlantic Ascension	20	2
7 Dutch Delight	50	3
8 Nordic Adventurer	40	[null]

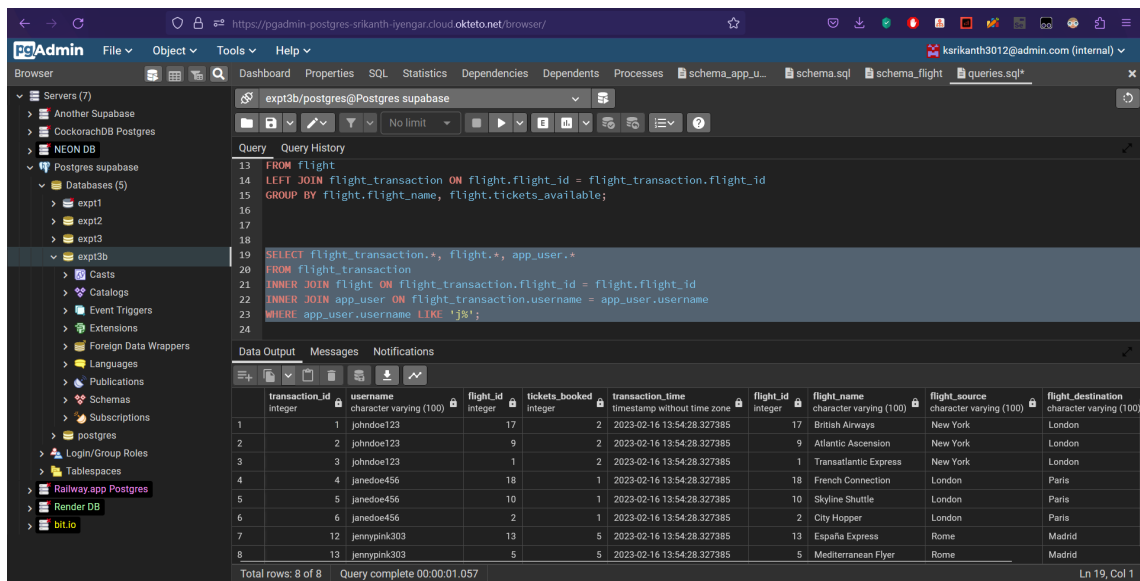
Total rows: 20 of 20 Query complete 00:00:01.800

Query 4

```
-- What are the flight details of all transactions made by users whose username
starts with the letter "J"?

SELECT flight_transaction.*, flight.*, app_user.*
FROM flight_transaction
INNER JOIN flight ON flight_transaction.flight_id = flight.flight_id
INNER JOIN app_user ON flight_transaction.username = app_user.username
WHERE app_user.username LIKE 'j%';
```

Result:



The screenshot shows the pgAdmin interface with a SQL query executed in the 'queries.sql*' tab. The query is a JOIN query that retrieves flight details, transaction information, and user details for users whose usernames start with 'J'. The results are displayed in a table with 8 rows and 9 columns.

transaction_id	username	flight_id	tickets_booked	transaction_time	flight_id	flight_name	flight_source	flight_destination
1	johndoe123	17	2	2023-02-16 13:54:28.327385	17	British Airways	New York	London
2	johndoe123	9	2	2023-02-16 13:54:28.327385	9	Atlantic Ascension	New York	London
3	johndoe123	1	2	2023-02-16 13:54:28.327385	1	Transatlantic Express	New York	London
4	janedoe456	18	1	2023-02-16 13:54:28.327385	18	French Connection	London	Paris
5	janedoe456	10	1	2023-02-16 13:54:28.327385	10	Skyline Shuttle	London	Paris
6	janedoe456	2	1	2023-02-16 13:54:28.327385	2	City Hopper	London	Paris
7	jennypink303	13	5	2023-02-16 13:54:28.327385	13	España Express	Rome	Madrid
8	jennypink303	5	5	2023-02-16 13:54:28.327385	5	Mediterranean Flyer	Rome	Madrid

Total rows: 8 of 8 Query complete 00:00:01.057 Ln 19, Col 1

Query 5

```
-- What are the top 5 flights with the highest number of tickets booked?

SELECT f.flight_name, COUNT(ft.tickets_booked) AS total_tickets_booked
FROM flight f
JOIN flight_transaction ft ON f.flight_id = ft.flight_id
GROUP BY f.flight_name
ORDER BY total_tickets_booked DESC
LIMIT 5;
```

Result:

The screenshot shows the pgAdmin web interface. On the left, a tree view displays the database structure, including 'Servers (7)', 'Databases (5)', and 'Postgres supabase'. The 'Postgres supabase' database is selected, showing its 'Databases (5)' and 'Schemas'.

The main pane shows a SQL query in the 'Query' tab:

```
21 INNER JOIN flight ON flight_transaction.flight_id = flight.flight_id
22 INNER JOIN app_user ON flight_transaction.username = app_user.username
23 WHERE app_user.username LIKE 'j%';
24
25
26
27 SELECT f.flight_name, COUNT(ft.tickets_booked) AS total_tickets_booked
28 FROM flight f
29 JOIN flight_transaction ft ON f.flight_id = ft.flight_id
30 GROUP BY f.flight_name
31 ORDER BY total_tickets_booked DESC
32 LIMIT 5;
```

The 'Data Output' tab shows the results of the query:

flight_name	total_tickets_booked
Iberian Explorer	1
European Voyager	1
German Getaway	1
Mediterranean Flyer	1
España Express	1

At the bottom, a status bar indicates: 'Total rows: 5 of 5', 'Query complete 00:00:02.263', and a green message: 'Successfully run. Total query runtime: 2 secs 263 msec. 5 rows affected.'

Conclusion

Learned how to design a distributed database by applying the concept of vertical fragmentation in PostgreSQL.