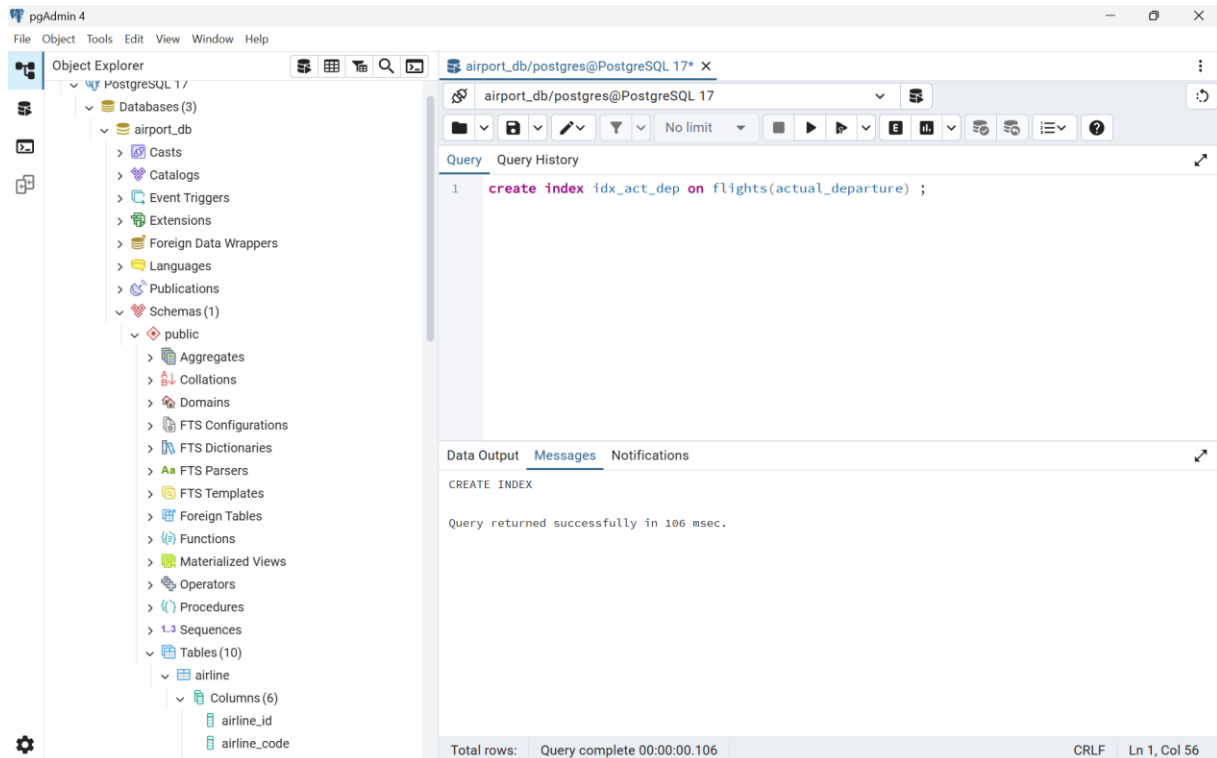
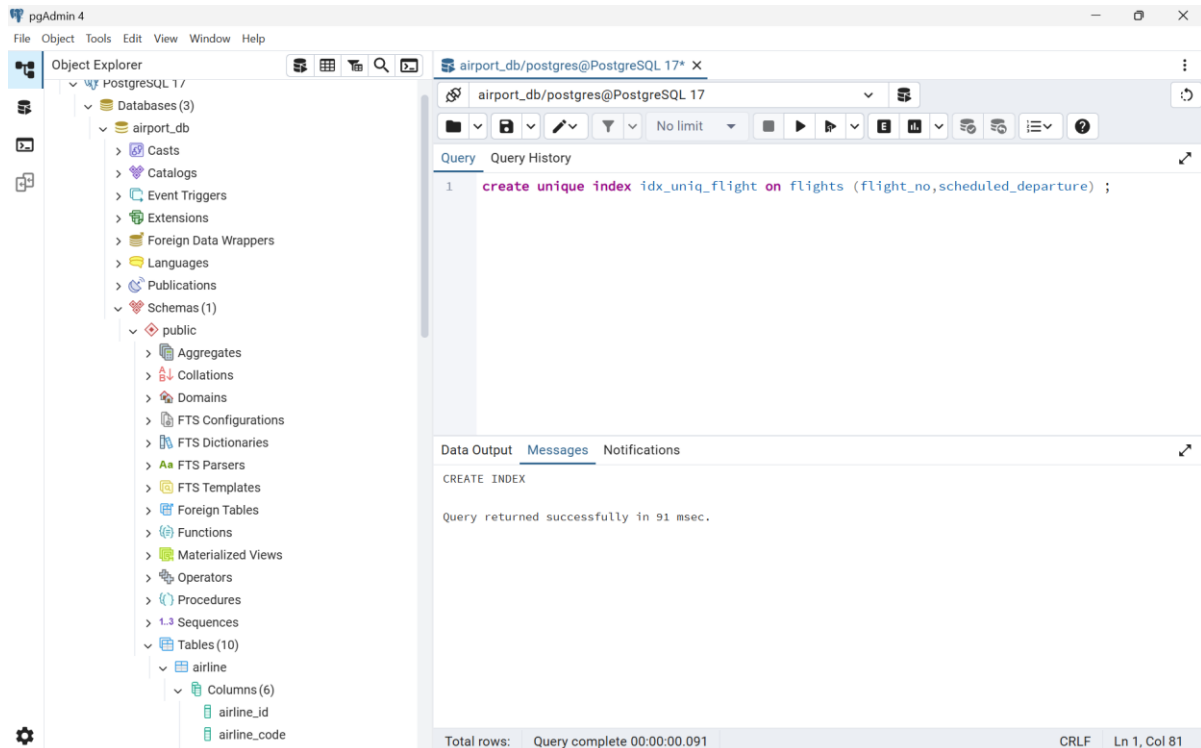


Laboratory work 7

1. Create an index on the actual_departure column in the flights table.



2. Create a unique index to ensure flight_no and scheduled_departure combinations are unique.



3. Create a composite index on the `departure_airport_id` and `arrival_airport_id` columns.

The screenshot shows the pgAdmin 4 interface. On the left, the Object Explorer displays the database structure: PostgreSQL 17 > Databases (3) > airport_db > Schemas (1) > public > Tables (10) > airline > Columns (6). The main query editor shows the following SQL command:

```
1 create index idx_airport on flights(departure_airport_id, arrival_airport_id);
```

The Messages pane at the bottom shows the execution result:

```
CREATE INDEX
Query returned successfully in 93 msec.
```

A green status bar at the bottom right confirms: "Query returned successfully in 93 msec." The status bar also indicates "Total rows: Query complete 00:00:00.093" and "CRLF Ln 1, Col 80".

4. Evaluate the difference in query performance with and without indexes. Measure performance differences.

The screenshot shows the pgAdmin 4 interface. On the left, the Object Explorer displays the database structure: PostgreSQL 17 > Databases (3) > airport_db > Schemas (1) > public > Tables (10) > flights > Columns (14). The main query editor shows the following SQL command:

```
1 explain analyze
2 select * from flights
3 where departure_airport_id = 3 and arrival_airport_id = 7;
```

The Messages pane at the bottom shows the execution result:

```
EXPLAIN ANALYZE
text
1 Seq Scan on flights (cost=0.00..36.00 rows=2 width=63) (actual time=0.280..0.394 rows=1 loops=...)
2 Filter: ((departure_airport_id = 3) AND (arrival_airport_id = 7))
3 Rows Removed by Filter: 999
4 Planning Time: 1.072 ms
5 Execution Time: 0.423 ms
```

The status bar at the bottom indicates "Total rows: 5" and "Query complete 00:00:00.201". The status bar also indicates "CRLF Ln 4, Col 1".

pgAdmin 4

File Object Tools Edit View Window Help

Object Explorer

- > Constraints
- > Indexes
- > RLS Policies
- > Rules
- > Triggers
- ▼ booking_flight
 - Columns (5)
 - booking_flight_id
 - booking_id
 - flight_id
 - created_at
 - update_at
 - > Constraints
 - > Indexes
 - > RLS Policies
 - > Rules
 - > Triggers
- ▼ flights
 - Columns (14)
 - flight_id
 - flight_no
 - scheduled_departure
 - scheduled_arrival
 - departure_airport_id
 - arrival_airport_id
 - departing_gate
 - arriving_gate
 - airline_id
 - status
 - actual_departure

airport_db/postgres@PostgreSQL 17*

airport_db/postgres@PostgreSQL 17

Query Query History

```
1 explain analyze
2 select * from flights
3 where departure_airport_id = 3 and arrival_airport_id = 7;
4
```

Data Output Messages Notifications

Showing rows: 1 to 7 Page No: 1 of 1

	QUERY PLAN
	text
1	Bitmap Heap Scan on flights (cost=4.30..10.47 rows=2 width=63) (actual time=0.401..0.402 rows=1 loops=1)
2	Recheck Cond: ((departure_airport_id = 3) AND (arrival_airport_id = 7))
3	Heap Blocks: exact=1
4	--> Bitmap Index Scan on idx_airport_pair (cost=0.00..4.29 rows=2 width=0) (actual time=0.364..0.364 rows=1 loop=1)
5	Index Cond: ((departure_airport_id = 3) AND (arrival_airport_id = 7))
6	Planning Time: 0.264 ms
7	Execution Time: 0.447 ms

Total rows: 7 Query complete 00:00:00.130 CRLF Ln 4, Col 1

5. Use EXPLAIN ANALYZE to check index usage in a query filtering by departure_airport and arrival_airport.

The screenshot shows the pgAdmin 4 interface. On the left, the Object Explorer displays the database structure for 'airport_db', including schemas, tables, and columns. The 'airline' table is expanded, showing columns 'airline_id' and 'airline_code'. The main query editor on the right contains the following SQL query:

```
1 explain analyze
2 select * from flights where departure_airport_id = 1 and arrival_airport_id = 1;
```

The query has been executed, and the 'Data Output' tab shows the query plan. The plan details are as follows:

Step	Operation	Cost	Rows	Width	Actual Time	Actual Rows	Actual Width	Loops
1	Bitmap Heap Scan on flights	cost=4.31..12.95	rows=3	width=63	actual time=0.794..0.803	rows=3	width=63	loops=1
2	Recheck Cond: ((departure_airport_id = 1) AND (arrival_airport_id = 1))							
3	Heap Blocks: exact=3							
4	-> Bitmap Index Scan on idx_airport	cost=0.00..4.31	rows=3	width=0	actual time=0.126..0.126	rows=3	width=0	loops=1
5	Index Cond: ((departure_airport_id = 1) AND (arrival_airport_id = 1))							
6	Planning Time: 0.326 ms							
7	Execution Time: 2.658 ms							

The status bar at the bottom indicates 'Total rows: 7' and 'Query complete 00:00:00.120'.

6. Create a unique index for the passport_number of the Passengers table. Check if the index was created or not. Insert into the table two new passengers.

Explain in your own words what is going on in the output?

create

The screenshot shows the pgAdmin 4 interface. The query editor contains the following SQL command:

```
1 create unique index idx_passport_un on passengers(passport_number);
```

The 'Execute script' button (labeled 'FS') has been clicked. The 'Data Output' tab shows the following message:

```
CREATE INDEX
Query returned successfully in 132 msec.
```

A green status bar at the bottom right confirms: 'Query returned successfully in 132 msec.' The status bar also shows 'Total rows: 0' and 'Query complete 00:00:00.132'.

Check

The screenshot shows the pgAdmin 4 interface. On the left, the Object Explorer displays the database structure for 'airport_db', including schemas, tables, and indexes. The main query editor on the right contains the following SQL query:

```
1 select indexname, indexdef from pg_indexes where tablename = 'passengers' ;
```

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

	indexname name	indexdef text
1	passengers_pkey	CREATE UNIQUE INDEX passengers_pkey ON public.passengers USING btree (passenger_id)
2	uq_passport_number	CREATE UNIQUE INDEX uq_passport_number ON public.passengers USING btree (passport_number)
3	idx_passport_uni	CREATE UNIQUE INDEX idx_passport_uni ON public.passengers USING btree (passport_number)

The status bar at the bottom indicates 'Total rows: 3' and 'Query complete 00:00:00.408'.

Insert 2 passengers

The screenshot shows the pgAdmin 4 interface. The main query editor contains the following SQL query:

```
1 insert into passengers
2 (passenger_id, first_name, last_name, date_of_birth, gender, country_of_citizenship,
3 country_of_residence, passport_number, created_at, update_at)
4 values
5 (201, 'Kuralay', 'Kassym', '2001-06-15', 'female', 'Kazakhstan', 'Kazakhstan', '4657385', '2
6 (202, 'Dana', 'Kairat', '1998-02-20', 'female', 'Kazakhstan', 'Kazakhstan', '4657385', '2024
7
```

The 'Messages' tab shows the following error message:

```
ERROR: повторяющееся значение ключа нарушает ограничение уникальности "uq_passport_number"
Ключ "(passport_number)=(4657385)" уже существует.

ОШИБКА: повторяющееся значение ключа нарушает ограничение уникальности "uq_passport_number"
SQL state: 23505
Detail: Ключ "(passport_number)=(4657385)" уже существует.
```

The status bar at the bottom indicates 'Total rows: 200' and 'Query complete 00:00:00.117'.

The unique index allows only one unique passport number per passenger,when I inserted 2 new passengers,it gave an error,because this 2 passengers have the same passport number. This happens because unique index prevents duplicate passport numbers in the table.

7. Create an index for the Passengers table. Use for that first name, last name, date of birth and country of citizenship. Then, write a SQL query to find a passenger who was born in Philippines and was born in 1984 and check if the query uses indexes or not. Give the explanation of the results.

The screenshot displays the pgAdmin 4 interface. On the left, the Object Explorer shows the database structure: Servers (1) > PostgreSQL 17 > Databases (3) > airport_db > Schemas (1) > public > Tables (10) > airline > Columns (6) > airline_id. The main pane shows the 'Query' tab with the following SQL query:

```
1 CREATE INDEX idx_passengerinfo ON passengers(first_name,last_name,date_of_birth,country_of_citizenship)
```

The 'Messages' tab shows the execution result:

```
CREATE INDEX
Query returned successfully in 109 msec.
```

A green status bar at the bottom right indicates: '✓ Query returned successfully in 109 msec. ✕'. The bottom status bar shows 'Total rows: Query complete 00:00:00.109' and 'CRLF Ln 1, Col 92'.

The screenshot shows the pgAdmin 4 interface. On the left, the Object Explorer displays the database structure for 'airport_db' under 'PostgreSQL 17'. The 'public' schema is expanded, showing various objects including 'airline' and its columns. The main query editor on the right contains the following SQL query:

```
1 explain analyze
2 select * from passengers where country_of_citizenship = 'Philippines' and date_of_birth between
```

Below the query editor, the 'Data Output' tab is active, showing the 'QUERY PLAN' for the executed query. The plan details are as follows:

Step	Operation	Cost	Rows	Width	Actual Time	Actual Rows	Actual Width	Loops
1	Seq Scan on passengers	cost=0.00..6.50	rows=1	width=64	actual time=0.041..0.102	rows=1	width=64	loops=1
2	Filter: ((date_of_birth >= '1984-01-01':date) AND (date_of_birth <= '1984-12-31':date) AND ((country_of_citizenship)::text = 'Philippines':te_							
3	Rows Removed by Filter: 199							
4	Planning Time: 1.491 ms							
5	Execution Time: 0.132 ms							

The status bar at the bottom indicates 'Total rows: 5', 'Query complete 00:00:00.104', and 'CRLF Ln 2, Col 128'.

The query used seq scan , so it checked every row in the table to find this passengers. It didn't use index that was created,because index is used from left to right,starting with the first columns that were defined in the index,also the table is small,so it would be faster than using the index , to scan all rows.

8. Write a SQL query to list indexes for table Passengers. After delete the created indexes.

