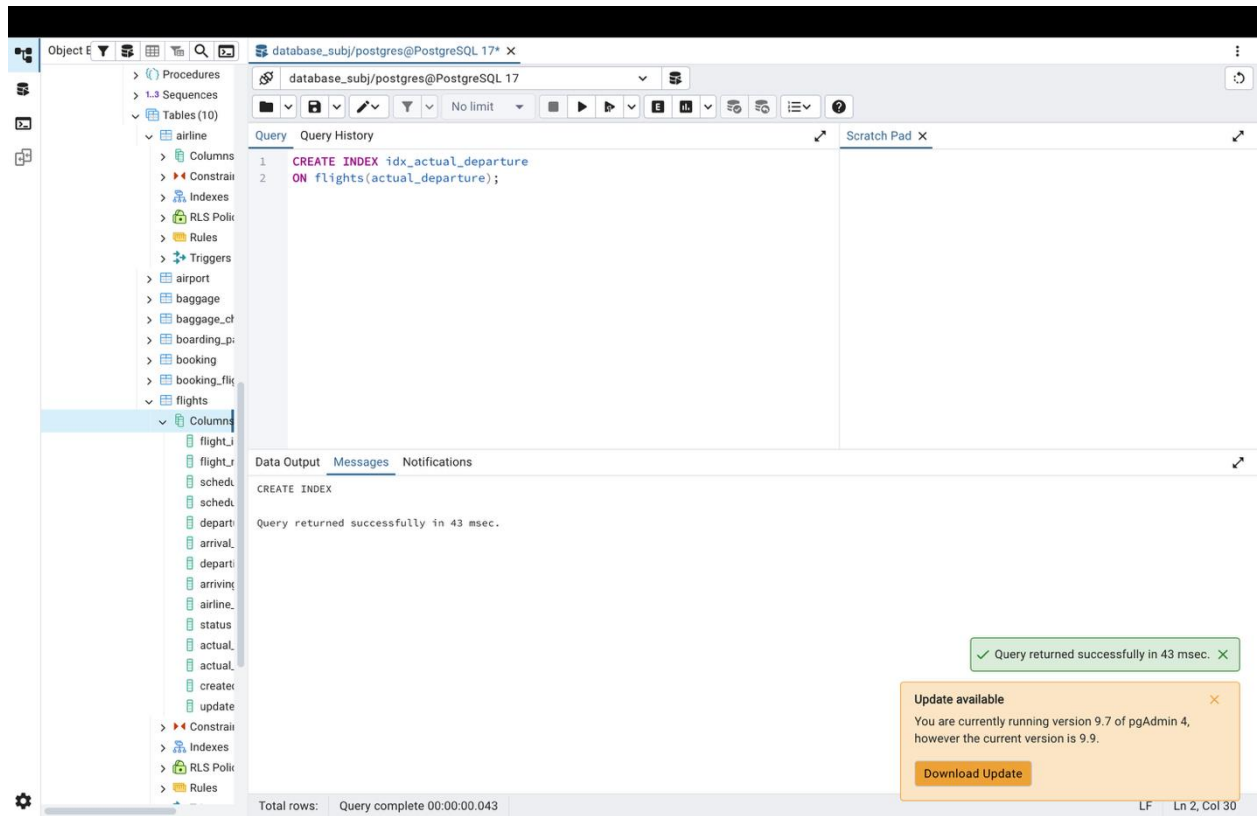


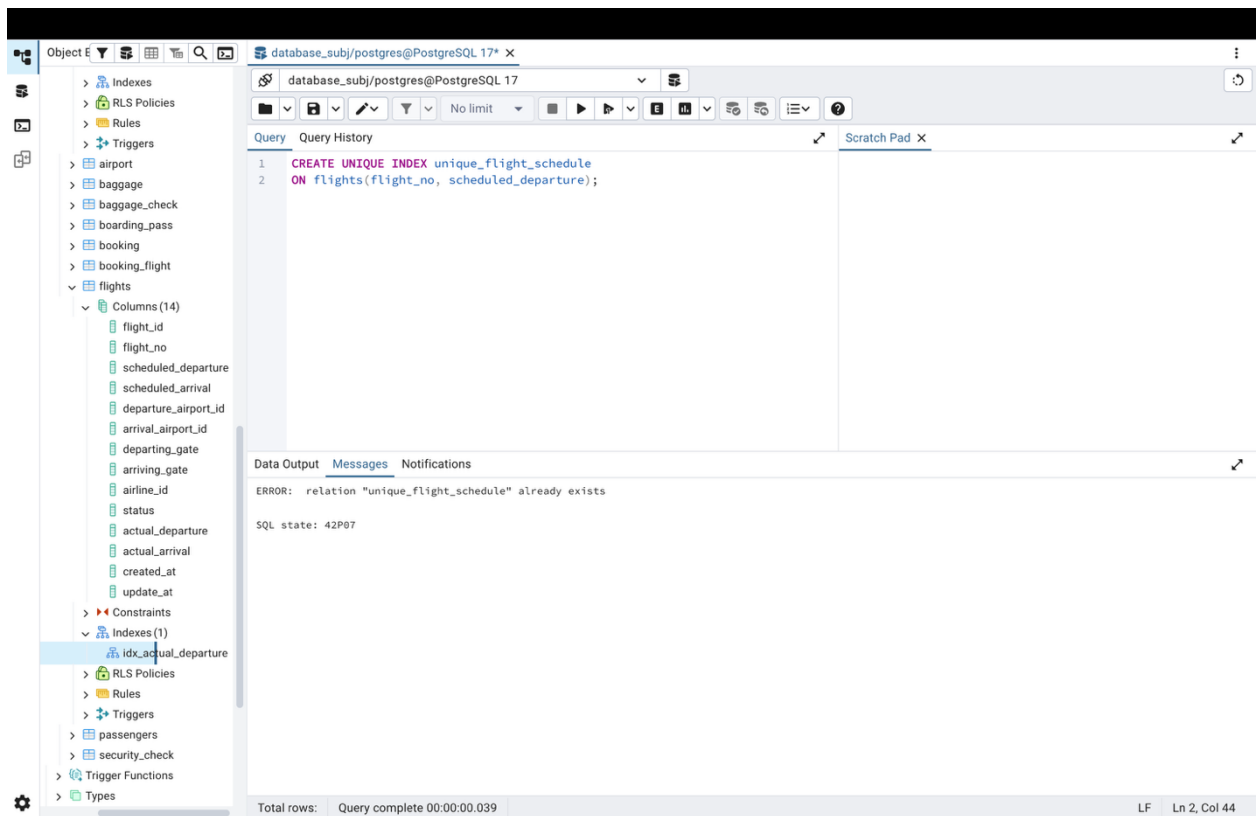
Laboratory work 7

Tasks:

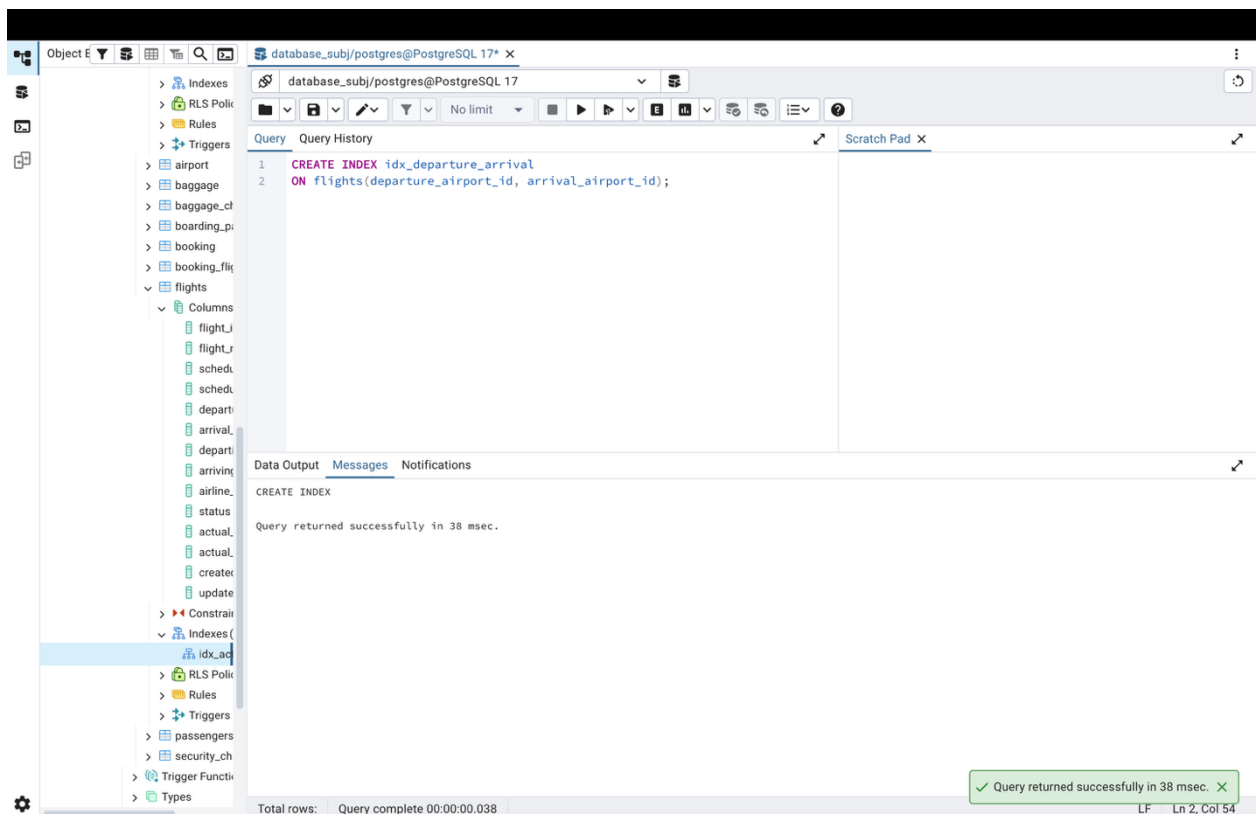
1. Create an index on the actual_departure column in the flights table.



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.



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

Object Explorer | database_subj/postgres@PostgreSQL 17*

Query | Query History | Scratch Pad

```

1 EXPLAIN ANALYZE
2 SELECT *
3 FROM flights
4 WHERE departure_airport_id = 1
5       AND arrival_airport_id = 2;

```

Data Output | Messages | Notifications

Showing rows: 1 to 5 | Page No: 1 | of 1

QUERY PLAN

Seq Scan on flights (cost=0.00..27.88 rows=2 width=61) (actual time=0.048..0.266 rows=2 loop=1)

Filter: ((departure_airport_id = 1) AND (arrival_airport_id = 2))

Rows Removed by Filter: 990

Planning Time: 0.486 ms

Execution Time: 0.295 ms

Total rows: 5 | Query complete 00:00:00.041

Successfully run. Total query runtime: 41 msec. 5 rows affected.

Object Explorer | database_subj/postgres@PostgreSQL 17*

Query | Query History | Scratch Pad

```

1 EXPLAIN ANALYZE
2 SELECT *
3 FROM flights
4 WHERE departure_airport_id = 1
5       AND arrival_airport_id = 2;

```

Data Output | Messages | Notifications

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

QUERY PLAN

Bitmap Heap Scan on flights (cost=4.30..9.97 rows=2 width=61) (actual time=0.064..0.067 rows=2 loops=1)

Recheck Cond: ((departure_airport_id = 1) AND (arrival_airport_id = 2))

Heap Blocks: exact=2

Bitmap Index Scan on idx_departure_arrival (cost=0.00..4.29 rows=2 width=0) (actual time=0.056..0.056 rows=2 loops=1)

Index Cond: ((departure_airport_id = 1) AND (arrival_airport_id = 2))

Planning Time: 0.775 ms

Execution Time: 0.115 ms

Total rows: 7 | Query complete 00:00:00.039

Successfully run. Total query runtime: 39 msec. 7 rows affected.

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

The screenshot shows the PostgreSQL IDE interface. On the left, the 'Object Explorer' pane displays the database schema, including tables like 'airport', 'baggage', 'boarding_pass', 'booking', 'booking_flight', and 'flights'. The 'flights' table is selected, showing its columns: flight_id, flight_no, scheduled_departure, scheduled_arrival, departure_airport_id, arrival_airport_id, departing_gate, arriving_gate, airline_id, status, actual_departure, actual_arrival, created_at, and update_at. The 'Indexes' section shows an index named 'idx_actual_departure'.

The main query window contains the following SQL query:

```

1 EXPLAIN ANALYZE
2 SELECT *
3 FROM flights
4 WHERE departure_airport_id = 1
5 AND arrival_airport_id = 2;

```

The 'Data Output' pane shows the 'QUERY PLAN' for the query:

```

1 Bitmap Heap Scan on flights (cost=4.30..9.97 rows=2 width=61) (actual time=0.038..0.042 rows=2 loops=1)
2   Recheck Cond: ((departure_airport_id = 1) AND (arrival_airport_id = 2))
3   Heap Blocks: exact=2
4   -> Bitmap Index Scan on idx_departure_arrival (cost=0.00..4.29 rows=2 width=0) (actual time=0.029..0.029 rows=2 loops=1)
5     Index Cond: ((departure_airport_id = 1) AND (arrival_airport_id = 2))
6 Planning Time: 0.190 ms
7 Execution Time: 0.079 ms

```

The status bar at the bottom indicates 'Total rows: 7' and 'Query complete 00:00:00.092'.

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?

The screenshot shows the PostgreSQL IDE interface with a new query window open. The query contains the following SQL commands:

```

1 -- In my own words: I created a unique index on passport_number to ensure each record is unique.
2 -- When inserting new passengers, PostgreSQL checks this index.
3 -- If the passport_number already exists, the insertion fails with a duplicate key error.
4 -- If the value is unique, the insertion succeeds.
5
6 -- Create a unique index on passport_number
7 CREATE UNIQUE INDEX IF NOT EXISTS unique_passport_numbers
8 ON passengers(passport_number);
9
10 -- Check that the index exists
11 SELECT indexname, indexdef
12 FROM pg_indexes
13 WHERE tablename = 'passengers';
14
15 -- Insert two new passengers
16 INSERT INTO passengers (
17     passenger_id, first_name, last_name, passport_number,
18     date_of_birth, gender, country_of_citizenship, country_of_residence,
19     created_at, update_at
20 )
21 VALUES
22 (201, 'John', 'Doe', 'P123456', '1990-01-01', 'M', 'USA', 'USA', CURRENT_DATE, CURRENT_DATE),
23 (202, 'Jane', 'Smith', 'P654321', '1992-02-02', 'F', 'USA', 'USA', CURRENT_DATE, CURRENT_DATE);

```

The 'Data Output' pane shows the result of the INSERT statement:

```

INSERT 0 2

```

The status bar at the bottom indicates 'Total rows: 2' and 'Query complete 00:00:00.031'.

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 shows the PostgreSQL IDE interface. On the left, the database schema for 'public.passengers' is visible, including columns like 'first_name', 'last_name', 'date_of_birth', and 'country_of_citizenship'. The main query editor contains the following SQL code:

```
1 -- Create a composite index
2 CREATE INDEX IF NOT EXISTS idx_passenger_info
3 ON passengers(first_name, last_name, date_of_birth, country_of_citizenship);
4
5 -- Query using a range to allow index usage
6 EXPLAIN ANALYZE
7 SELECT *
8 FROM passengers
9 WHERE date_of_birth BETWEEN '1984-01-01' AND '1984-12-31'
10 AND country_of_citizenship = 'Philippines';
11
12 -- Explanation:
13 -- The composite index on first_name, last_name, date_of_birth, and country_of_citizenship
14 -- helps PostgreSQL quickly locate rows matching the query.
15 -- We use a date range for date_of_birth instead of a function like date_part('year', ...)
16 -- because functions prevent the index from being used.
17 -- In the EXPLAIN ANALYZE output, if you see "Index Scan" or "Bitmap Index Scan",
18 -- it means PostgreSQL is using the index to fetch rows efficiently.
19 -- If you see "Seq Scan", the index is not being used, and PostgreSQL is scanning the whole table.
20 -- Using the index significantly reduces query time, especially on large tables.
```

The 'Data Output' tab shows the 'QUERY PLAN' for the executed query:

Step	Operation	Cost	Rows	Width	Actual Time	Actual Rows	Actual Width	Loops
1	Seq Scan on passengers	cost=0.00..6.54	rows=1	width=64	(actual time=0.020..0.042	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':text))							
3	Rows Removed by Filter: 201							
4	Planning Time: 1.469 ms							
5	Execution Time: 0.078 ms							

The status bar at the bottom indicates 'Query executed with server cursor' and 'Query complete 00:00:00.042'.

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

The screenshot shows the PostgreSQL IDE interface. The main query editor contains the following SQL code:

```
1 SELECT indexname, indexdef
2 FROM pg_indexes
3 WHERE tablename = 'passengers';
4
5 DROP INDEX IF EXISTS idx_passenger_info;
6
7 DROP INDEX IF EXISTS unique_passport_numbers;
```

The 'Data Output' tab shows the 'DROP INDEX' query result:

```
Query returned successfully in 46 msec.
```

The status bar at the bottom indicates 'Query executed with server cursor' and 'Query complete 00:00:00.046'. A green notification box at the bottom right says 'Query returned successfully in 46 msec.'.