

Laboratory work 7

Tasks:

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

The screenshot shows the pgAdmin 4 interface. On the left, the Object Explorer tree shows a database named 'database_subj' with a 'flights' table selected. Under the 'Columns' section of the flights table, there is a 'actual_departure' column. In the center, the Query Editor window contains the following SQL code:

```
CREATE INDEX idx_actual_departure
ON flights(actual_departure);
```

Below the code, the 'Messages' tab shows the output: 'CREATE INDEX' and 'Query returned successfully in 43 msec.'. A green message box at the bottom right indicates 'Query returned successfully in 43 msec.' with a checkmark icon. At the bottom of the interface, a status bar shows 'Total rows: 0' and 'Query complete 00:00:00.043'.

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

The screenshot shows the pgAdmin 4 interface. On the left, the Object Explorer displays the database schema, including tables like airport, baggage, boarding_pass, booking, booking_flight, and flights. The flights table has 14 columns. In the center, the Query Editor window contains the following SQL code:

```

CREATE UNIQUE INDEX unique_flight_schedule
ON flights(flight_no, scheduled_departure);

```

The Data Output tab shows the error message: "ERROR: relation "unique_flight_schedule" already exists". The status bar at the bottom indicates "Query complete 00:00:00.039".

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

The screenshot shows the pgAdmin 4 interface. The Object Explorer on the left shows the database schema. The Query Editor window contains the following SQL code:

```

CREATE INDEX idx_departure_arrival
ON flights(departure_airport_id, arrival_airport_id);

```

The Data Output tab shows the message: "CREATE INDEX" and "Query returned successfully in 38 msec.". A green success message at the bottom right says "✓ Query returned successfully in 38 msec. X". The status bar at the bottom indicates "Query complete 00:00:00.038".

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

```

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

```

QUERY PLAN

- Seq Scan on flights (cost=0.00..27.88 rows=2 width=61) (actual time=0.048..0.266 rows=2 loops=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 LF Ln 5, Col 30

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

```

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

```

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 LF Ln 5, Col 30

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 pgAdmin 4 interface with the following details:

- Object Explorer:** Shows the database structure with tables like `flights`, `airports`, and `passengers`.
- Query Editor:** Contains the following SQL code:


```
EXPLAIN ANALYZE
SELECT *
FROM flights
WHERE departure_airport_id = 1
  AND arrival_airport_id = 2;
```
- Data Output / Query Plan:** Displays the execution plan:


```
QUERY PLAN
text
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
```
- Status:** Total rows: 7 | Query complete 00:00:00.092 | LF | Ln 5, Col 30

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 pgAdmin 4 interface with the following details:

- Object Explorer:** Shows the database structure with the `passengers` table.
- Query Editor:** Contains the following SQL code:


```
-- In my own words: I created a unique index on passport_number to ensure each record is unique.
-- When inserting new passengers, PostgreSQL checks this index.
-- If the passport_number already exists, the insertion fails with a duplicate key error.
-- If the value is unique, the insertion succeeds.

-- Create a unique index on passport_number
CREATE UNIQUE INDEX IF NOT EXISTS unique_passport_numbers
ON passengers(passport_number);

-- Check that the index exists
SELECT indexname, indexdef
FROM pg_indexes
WHERE tablename = 'passengers';

-- Insert two new passengers
INSERT INTO passengers (
  passenger_id, first_name, last_name, passport_number,
  date_of_birth, gender, country_of_citizenship, country_of_residence,
  created_at, update_at
)
VALUES
(201, 'John', 'Doe', 'P123456', '1990-01-01', 'M', 'USA', 'USA', CURRENT_DATE, CURRENT_DATE),
(202, 'Jane', 'Smith', 'P654321', '1992-02-02', 'F', 'USA', 'USA', CURRENT_DATE, CURRENT_DATE);
```
- Data Output:** Shows the results of the insert operation:


```
INSERT 0 2
Query returned successfully in 31 msec.
```
- Status:** Total rows: 0 | Query complete 00:00:00.031 | LF | Ln 10, Col 31

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.

```

CREATE INDEX IF NOT EXISTS idx_passenger_info
ON passengers(first_name, last_name, date_of_birth, country_of_citizenship);

EXPLAIN ANALYZE
SELECT *
FROM passengers
WHERE date_of_birth BETWEEN '1984-01-01' AND '1984-12-31'
    AND country_of_citizenship = 'Philippines';

-- Explanation:
-- The composite index on first_name, last_name, date_of_birth, and country_of_citizenship
-- helps PostgreSQL quickly locate rows matching the query.
-- We use a date range for date_of_birth instead of function like date_part('year', ...)
-- because functions prevent the index from being used.
-- In the EXPLAIN ANALYZE output, if you see "Index Scan" or "Bitmap Index Scan",
-- it means PostgreSQL is using the index to fetch rows efficiently.
-- If you see "Seq Scan", the index is not being used, and PostgreSQL is scanning the whole table.
-- Using the index significantly reduces query time, especially on large tables.

```

Query executed with server cursor Query complete 00:00:00.042 LF Ln 20, Col 81

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

```

SELECT indexname, indexdef
FROM pg_indexes
WHERE tablename = 'passenger';

DROP INDEX IF EXISTS idx_passenger_info;
DROP INDEX IF EXISTS unique_passport_numbers;

```

Data Output Messages Notifications

DROP INDEX

Query returned successfully in 46 msec.

✓ Query returned successfully in 46 msec. X

Query executed with server cursor Query complete 00:00:00.046 LF Ln 7, Col 45