

Laboratory work 7

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 displays various database objects like Collations, Domains, FTS Configurations, etc., under the 'Tables (10)' section, where the 'flights' table is expanded to show its 14 columns. In the center, the Query Editor window contains the SQL command: `CREATE INDEX idx_actual_departure ON flights(actual_departure);`. Below the query, the 'Messages' tab shows the response: `CREATE INDEX` and `Query returned successfully in 63 msec.`. At the bottom, a status bar indicates `Total rows: 0`, `Query complete 00:00:00.063`, and a green message box says `✓ Query returned successfully in 63 msec. X`. The top navigation bar shows the connection details: LABWORKS/postgres@PostgreSQL 17.

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 pane displays a tree view of database objects, including tables like airline, airport, baggage, etc., and a detailed view of the flights table with its 14 columns. In the center, the main window shows a query editor with the following SQL command:

```
CREATE UNIQUE INDEX idx_flights_no_sched_departure ON flights (flight_no, scheduled_departure);
```

The 'Messages' tab in the query editor shows the response:

```
CREATE INDEX
Query returned successfully in 72 msec.
```

A green status bar at the bottom right indicates:

✓ Query returned successfully in 72 msec. X
LF Ln 1, Col 52

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 pane on the left shows the same tree structure as the previous screenshot. In the center, the query editor contains the following SQL command:

```
CREATE INDEX idx_flights_departure_arrival ON flights (departure_airport_id, arrival_airport_id);
```

The 'Messages' tab shows the response:

```
CREATE INDEX
Query returned successfully in 58 msec.
```

A green status bar at the bottom right indicates:

✓ Query returned successfully in 58 msec. X
LF Ln 1, Col 44

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

The screenshot shows the pgAdmin 4 interface with two query panes. The left pane displays the Object Explorer with the 'flights' table selected. The right pane contains two queries:

```

1 SET enable_seqscan = ON;
2 v EXPLAIN ANALYZE
3 SELECT * FROM flights WHERE departure_airport_id = 1 AND arrival_airport_id = 2;
4
5 SET enable_seqscan = OFF;
6 v EXPLAIN ANALYZE
7 SELECT * FROM flights WHERE departure_airport_id = 1 AND arrival_airport_id = 2;

```

The 'Data Output' tab shows the execution plan for both queries. The first query (with indexes) uses a Bitmap Index Scan on the idx_flights_departure_arrival index, while the second query (without indexes) uses a Bitmap Heap Scan on the flights table. The execution times are 0.047ms and 0.069ms respectively.

The screenshot shows the pgAdmin 4 interface with two query panes. The left pane displays the Object Explorer with the 'flights' table selected. The right pane contains two queries:

```

1 SET enable_seqscan = ON;
2 v EXPLAIN ANALYZE
3 SELECT * FROM flights WHERE departure_airport_id = 1 AND arrival_airport_id = 2;
4
5 SET enable_seqscan = OFF;
6 v EXPLAIN ANALYZE
7 SELECT * FROM flights WHERE departure_airport_id = 1 AND arrival_airport_id = 2;

```

The 'Data Output' tab shows the execution plan for both queries. The first query (with indexes) uses a Bitmap Index Scan on the idx_flights_departure_arrival index, while the second query (without indexes) uses a Bitmap Heap Scan on the flights table. The execution times are 0.031ms and 0.089ms respectively.

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 is the Object Explorer tree, which includes sections like Collations, Domains, FTS Configurations, FTS Dictionaries, FTS Parsers, FTS Templates, Foreign Tables, Functions, Materialized Views, Operators, Procedures, Sequences, and Tables (10). Under Tables, the 'flights' table is selected, showing its 14 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. Below the tree are sections for Constraints, Indexes, RLS Policies, Rules, and Triggers, along with a 'nasspaaare' entry.

The main area contains a query editor window titled 'LABWORKS/postgres@PostgreSQL 17'. It displays the following SQL command:

```
EXPLAIN ANALYZE
SELECT * FROM flights WHERE departure_airport_id = 3 AND arrival_airport_id = 7;
```

Below the query is a 'Data Output' tab showing the 'QUERY PLAN' text:

```
text
1 Bitmap Heap Scan on flights (cost=4.30..9.97 rows=2 width=61) (actual time=0.061..0.061 rows=1 loops=1)
  Recheck Cond: ((departure_airport_id = 3) AND (arrival_airport_id = 7))
  Heap Blocks: exact=1
  -> Bitmap Index Scan on idx_flights_departure_arrival (cost=0.00..4.29 rows=2 width=0) (actual time=0.041..0.042 rows=1 loops=1)
    Index Cond: ((departure_airport_id = 3) AND (arrival_airport_id = 7))
Planning Time: 0.173 ms
Execution Time: 0.100 ms
```

The status bar at the bottom indicates 'Total rows: 7' and 'Query complete 00:00:00.102'. A green message box says 'Successfully run. Total query runtime: 102 msec. 7 rows affected.' with a close button.

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?

Object Explorer

LABWORKS/postgres@PostgreSQL 17

```

CREATE UNIQUE INDEX idx_passengers_passport_number ON passengers (passport_number);

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

INSERT INTO passengers (passenger_id, first_name, last_name, passport_number)
VALUES (1001, 'Lia', 'Park', 'P123456');

INSERT INTO passengers (passenger_id, first_name, last_name, passport_number)
VALUES (1002, 'Emma', 'Kim', 'P123456');

```

Data Output Messages Notifications

CREATE INDEX

Query returned successfully in 92 msec.

Total rows: Query complete 00:00:00.092

✓ Query returned successfully in 92 msec. LF Ln 1, Col 1

Object Explorer

LABWORKS/postgres@PostgreSQL 17

```

CREATE UNIQUE INDEX idx_passengers_passport_number ON passengers (passport_number);

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

INSERT INTO passengers (passenger_id, first_name, last_name, passport_number)
VALUES (1001, 'Lia', 'Park', 'P123456');

INSERT INTO passengers (passenger_id, first_name, last_name, passport_number)
VALUES (1002, 'Emma', 'Kim', 'P123456');

```

Data Output Messages Notifications

indexname	indexdef
passengers_pkey	CREATE UNIQUE INDEX passengers_pkey ON public.passengers USING btree (passenger_id)
idx_passengers_passport_number	CREATE UNIQUE INDEX idx_passengers_passport_number ON public.passengers USING btree (passport_number)

Showing rows: 1 to 2 Page No: 1 of 1 << <> >> >>

Total rows: 2 Query complete 00:00:00.076

✓ Successfully run. Total query runtime: 76 msec. 2 rows affected. LF Ln 3, Col 1

The screenshot shows the pgAdmin 4 interface with the Object Explorer on the left and a query editor on the right. The Object Explorer displays a tree structure of database objects, including tables like `airline`, `airport`, `baggage`, etc., and the `flights` table which has 14 columns. The `passengers` table is also listed. The query editor contains the following SQL code:

```
CREATE UNIQUE INDEX idx_passengers_passport_number ON passengers (passport_number);
SELECT indexname, indexdef
FROM pg_indexes
WHERE tablename = 'passengers';
INSERT INTO passengers (passenger_id, first_name, last_name, passport_number)
VALUES (1001, 'Lia', 'Park', 'P123456');
INSERT INTO passengers (passenger_id, first_name, last_name, passport_number)
VALUES (1002, 'Emma', 'Kim', 'P123456');
```

The Data Output tab shows the results of the second `INSERT` statement:

```
INSERT 0 1
```

Query returned successfully in 67 msec.

Total rows: Query complete 00:00:00.067

LF Ln 10, Col 1

The screenshot shows the pgAdmin 4 interface with the Object Explorer on the left and a query editor on the right. The Object Explorer displays the same tree structure as the previous screenshot. The query editor contains the same SQL code as the previous screenshot, but the execution results show an error:

```
CREATE UNIQUE INDEX idx_passengers_passport_number ON passengers (passport_number);
SELECT indexname, indexdef
FROM pg_indexes
WHERE tablename = 'passengers';
INSERT INTO passengers (passenger_id, first_name, last_name, passport_number)
VALUES (1001, 'Lia', 'Park', 'P123456');
INSERT INTO passengers (passenger_id, first_name, last_name, passport_number)
VALUES (1002, 'Emma', 'Kim', 'P123456');
```

The Data Output tab shows the error message:

```
ERROR: duplicate key value violates unique constraint "idx_passengers_passport_number"
Key (passport_number)=(P123456) already exists.
```

SQL state: 23505

Detail: Key (passport_number)=(P123456) already exists.

Total rows: Query complete 00:00:00.055

LF Ln 10, Col 1

The error means you tried to insert a new passenger with the same passport number as one that already exists.

Because the column has a unique index, PostgreSQL doesn't allow duplicate passport numbers, so it stops the insertion and shows this error.

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 pgAdmin 4 interface. On the left, the Object Explorer tree displays various database objects like Collations, Domains, FTS Configurations, etc., under the 'Tables (10)' section, which includes 'flights'. Under 'flights', the 'Columns (14)' section is expanded, showing columns such as flight_id, flight_no, scheduled_departure, and others. In the center, the Query History tab contains the SQL command:

```
CREATE INDEX idx_passengers_name_dob_country ON passengers (first_name, last_name, date_of_birth, country_of_citizenship);
```

Below the query, the Data Output tab shows the message "CREATE INDEX" and "Query returned successfully in 39 msec." At the bottom, status information indicates "Total rows: 0" and "Query complete 00:00:00.039". A green success message box at the bottom right states "✓ Query returned successfully in 39 msec. LF Ln 1, Col 46".

The screenshot shows the pgAdmin 4 interface. The left sidebar is the Object Explorer, listing various database objects like Collations, Domains, FTS Configurations, etc., under the schema LABWORKS/postgres@PostgreSQL 17*. The main area is the Query History tab, which contains the following SQL code:

```

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

```

Below the code, the Data Output tab shows the results of the query. The Query Plan section indicates an Index Scan using idx_passengers_name_dob_country on the passengers table. The execution time was 0.839 ms.

Total rows: 4 Query complete 00:00:00.058 ✓ Successfully run. Total query runtime: 58 msec. 4 rows affected. LF Ln 2, Col 71

PostgreSQL used the index to find passengers from the Philippines born in 1984. Because of the index, it didn't need to scan the whole table, so the query ran much faster, less than 1 millisecond.

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

Object Explorer

Collations
Domains
FTS Configurations
FTS Dictionaries
FTS Parsers
FTS Templates
Foreign Tables
Functions
Materialized Views
Operators
Procedures
Sequences
Tables (10)
airline
airport
baggage
baggage_check
boarding_pass
booking
booking_flight
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
actual_arrival
created_at
update_at

Constraints
Indexes
RLS Policies
Rules
Triggers

passengers

LABWORKS/postgres@PostgreSQL 17

No limit

Query History

```
1 SELECT indexname, indexdef FROM pg_indexes WHERE tablename = 'passengers';
2
3 DROP INDEX IF EXISTS idx_passengers_passport_number;
4 DROP INDEX IF EXISTS idx_passengers_name_dob_country;
```

Data Output Messages Notifications

Showing rows: 1 to 3 Page No: 1 of 1

indexname	indexdef
passengers_pkey	CREATE UNIQUE INDEX passengers_pkey ON public.passengers USING btree (passenger_id)
idx_passengers_passport_number	CREATE UNIQUE INDEX idx_passengers_passport_number ON public.passengers USING btree (passport_number)
idx_passengers_name_dob_country	CREATE INDEX idx_passengers_name_dob_country ON public.passengers USING btree (first_name, last_name, date_of_birth, country_of_citizenship)

Total rows: 3 Query complete 00:00:00.075

Successfully run. Total query runtime: 75 msec. 3 rows affected.

The screenshot shows the DBeaver interface with the following details:

- Object Explorer:** On the left, it lists database objects under "Tables (10)". The "flights" table is expanded, 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.
- Query Editor:** The main area contains a query window titled "LABWORKS/postgres@PostgreSQL 17". The query is:

```
1 SELECT indexname, indexdef FROM pg_indexes WHERE tablename = 'passengers';
2
3 DROP INDEX IF EXISTS idx_passengers_passport_number;
4 DROP INDEX IF EXISTS idx_passengers_name_dob_country;
```
- Data Output:** Below the query window, the "Data Output" tab is active, showing the message: "Query returned successfully in 56 msec."
- Messages:** The "Messages" tab shows the command: "DROP INDEX" followed by the message: "DROP INDEX".
- Status Bar:** At the bottom right, a green message box indicates: "✓ Query returned successfully in 56 msec. Lf Ln 3, Col 1".