

## Laboratory work 7

1. Create an index on the actual\_departure column in the flights table.

The screenshot displays the PostgreSQL Enterprise Studio interface. On the left, the Object Explorer shows the database structure, with the 'actual\_departure' column in the 'flights' table selected. The main query editor contains the following SQL statement:

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

The 'Data Output' tab at the bottom shows the execution results:

```
CREATE INDEX  
Query returned successfully in 63 msec.
```

A status bar at the bottom right indicates: 'Query returned successfully in 63 msec.' and 'Total rows: Query complete 00:00:00.063'.

2. Create a unique index to ensure flight\_no and scheduled\_departure combinations are unique.

The screenshot shows the SQL Server Enterprise Manager interface. On the left, the Object Explorer displays the database structure, with the 'flights' table selected under 'Tables (10)'. The 'Columns (14)' list for the 'flights' table is expanded, showing columns like 'flight\_id', 'flight\_no', 'scheduled\_departure', 'scheduled\_arrival', 'departure\_airport\_id', and 'arrival\_airport\_id'. The 'scheduled\_departure' column is highlighted. The main pane shows a SQL query in the 'Query' tab:

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

The 'Messages' tab at the bottom shows the execution result: 'CREATE INDEX' and 'Query returned successfully in 72 msec.' A status bar at the bottom indicates 'Total rows: Query complete 00:00:00.072'.

3. Create a composite index on the departure\_airport\_id and arrival\_airport\_id columns.

The screenshot shows the SQL Server Enterprise Manager interface. On the left, the Object Explorer displays the database structure, with the 'flights' table selected under 'Tables (10)'. The 'Columns (14)' list for the 'flights' table is expanded, showing columns like 'flight\_id', 'flight\_no', 'scheduled\_departure', 'scheduled\_arrival', 'departure\_airport\_id', and 'arrival\_airport\_id'. The 'departure\_airport\_id' column is highlighted. The main pane shows a SQL query in the 'Query' tab:

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

The 'Messages' tab at the bottom shows the execution result: 'CREATE INDEX' and 'Query returned successfully in 58 msec.' A status bar at the bottom indicates 'Total rows: Query complete 00:00:00.058'.

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

The screenshot shows the PostgreSQL IDE interface. On the left, the 'Object Explorer' pane displays a tree view of the database schema, including tables like 'airline', 'airport', 'baggage', 'boarding\_pass', 'booking', 'booking\_flight', and 'flights'. The 'flights' table is selected, and its columns are listed: '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 main query editor displays the following SQL script:

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

The 'Data Output' pane shows the 'QUERY PLAN' for the first query (with seqscan ON):

Step	Operation	Cost	Actual Time	Actual Rows	Actual Loops
1	Bitmap Heap Scan on flights	(cost=4.30..9.97 rows=2 width=61)	0.062	0.069	2
2	Recheck Cond: ((departure_airport_id = 1) AND (arrival_airport_id = 2))				
3	Heap Blocks: exact=2				
4	Bitmap Index Scan on idx_flights_departure_arrival	(cost=0.00..4.29 rows=2 width=0)	0.047	0.047	2
5	Index Cond: ((departure_airport_id = 1) AND (arrival_airport_id = 2))				
6	Planning Time	0.129 ms			
7	Execution Time	0.094 ms			

The status bar at the bottom indicates 'Total rows: 7' and 'Query complete 00:00:00.066'. A green message box at the bottom right states: 'Successfully run. Total query runtime: 66 msec. 7 rows affected.'

The screenshot shows the PostgreSQL IDE interface, similar to the first one, but with the second query (with seqscan OFF) selected in the query editor. The 'Data Output' pane shows the 'QUERY PLAN' for the second query (with seqscan OFF):

Step	Operation	Cost	Actual Time	Actual Rows	Actual Loops
1	Bitmap Heap Scan on flights	(cost=4.30..9.97 rows=2 width=61)	0.042	0.046	2
2	Recheck Cond: ((departure_airport_id = 1) AND (arrival_airport_id = 2))				
3	Heap Blocks: exact=2				
4	Bitmap Index Scan on idx_flights_departure_arrival	(cost=0.00..4.29 rows=2 width=0)	0.031	0.031	2
5	Index Cond: ((departure_airport_id = 1) AND (arrival_airport_id = 2))				
6	Planning Time	0.080 ms			
7	Execution Time	0.060 ms			

The status bar at the bottom indicates 'Total rows: 7' and 'Query complete 00:00:00.089'. A green message box at the bottom right states: 'Successfully run. Total query runtime: 89 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 a PostgreSQL IDE interface. On the left, the 'Object Explorer' pane displays the database schema, including tables like 'airline', 'airport', 'baggage', 'boarding\_pass', 'booking', 'booking\_flight', and 'flights'. The 'flights' table is selected, and its columns are listed: 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 main query editor displays the following SQL query:

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

The 'Data Output' pane shows the query plan:

```
QUERY PLAN
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)
2    Recheck Cond: ((departure_airport_id = 3) AND (arrival_airport_id = 7))
3    Heap Blocks: exact=1
4    -> 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 loop=1)
5          Index Cond: ((departure_airport_id = 3) AND (arrival_airport_id = 7))
6    Planning Time: 0.173 ms
7    Execution Time: 0.100 ms
```

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

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

- 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

Query

```
1 CREATE UNIQUE INDEX idx_passengers_passport_number ON passengers (passport_number);
2
3 SELECT indexname, indexdef
4 FROM pg_indexes
5 WHERE tablename = 'passengers';
6
7 INSERT INTO passengers (passenger_id, first_name, last_name, passport_number)
8 VALUES (1001, 'Lia', 'Park', 'P123456');
9
10 INSERT INTO passengers (passenger_id, first_name, last_name, passport_number)
11 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

- 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

Query

```
1 CREATE UNIQUE INDEX idx_passengers_passport_number ON passengers (passport_number);
2
3 SELECT indexname, indexdef
4 FROM pg_indexes
5 WHERE tablename = 'passengers';
6
7 INSERT INTO passengers (passenger_id, first_name, last_name, passport_number)
8 VALUES (1001, 'Lia', 'Park', 'P123456');
9
10 INSERT INTO passengers (passenger_id, first_name, last_name, passport_number)
11 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 SQL Studio interface with the 'Object Explorer' on the left and the 'Query' editor on the right. The 'Object Explorer' shows a tree view of the database schema, including tables, columns, constraints, and indexes. The 'Query' editor contains the following SQL code:

```
1 CREATE UNIQUE INDEX idx_passengers_passport_number ON passengers (passport_number);
2
3 SELECT indexname, indexdef
4 FROM pg_indexes
5 WHERE tablename = 'passengers';
6
7 INSERT INTO passengers (passenger_id, first_name, last_name, passport_number)
8 VALUES (1001, 'Lia', 'Park', 'P123456');
9
10 INSERT INTO passengers (passenger_id, first_name, last_name, passport_number)
11 VALUES (1002, 'Emma', 'Kim', 'P123456');
```

The 'Data Output' tab shows the results of the query, indicating that the query was successful and returned 0 rows. A green message box at the bottom right states: "Query returned successfully in 67 msec."

The screenshot shows the SQL Studio interface with the 'Object Explorer' on the left and the 'Query' editor on the right. The 'Object Explorer' shows a tree view of the database schema, including tables, columns, constraints, and indexes. The 'Query' editor contains the following SQL code:

```
1 CREATE UNIQUE INDEX idx_passengers_passport_number ON passengers (passport_number);
2
3 SELECT indexname, indexdef
4 FROM pg_indexes
5 WHERE tablename = 'passengers';
6
7 INSERT INTO passengers (passenger_id, first_name, last_name, passport_number)
8 VALUES (1001, 'Lia', 'Park', 'P123456');
9
10 INSERT INTO passengers (passenger_id, first_name, last_name, passport_number)
11 VALUES (1002, 'Emma', 'Kim', 'P123456');
```

The 'Data Output' tab shows the results of the query, indicating that the query failed due to a duplicate key error. The error message is: "ERROR: duplicate key value violates unique constraint \"idx\_passengers\_passport\_number\" Key (passport\_number)=(P123456) already exists." The SQL state is 23505. The detail is: "Key (passport\_number)=(P123456) already exists."

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 displays the PostgreSQL Labworks interface. On the left, the Object Explorer shows the database structure, including tables and columns. The 'passengers' table is selected, and its columns are listed: 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 'Columns (14)' section is expanded, and 'scheduled\_departure' is highlighted. The main query editor shows the following SQL command:

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

The 'Data Output' tab is active, showing the message: 'CREATE INDEX' and 'Query returned successfully in 39 msec.' A status bar at the bottom indicates 'Total rows: Query complete 00:00:00.039' and 'LF Ln 1, Col 46'.

The screenshot displays the PostgreSQL Enterprise Studio interface. On the left, the 'Object Explorer' pane shows the database schema, with the 'passengers' table selected under the 'flights' schema. The 'Columns (14)' list for the 'passengers' table is visible, including '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 is also visible, showing 'idx\_passengers\_name\_dob\_country'.

The main query editor shows the following SQL query:

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

The 'Data Output' pane shows the 'QUERY PLAN' results:

Step	Text
1	Index Scan using idx_passengers_name_dob_country on passengers (cost=0.27..22.79 rows=1 width=64) (actual time=0.251..0.299 rows=1 loop=1)
2	Index Cond: ((date_of_birth >= '1984-01-01':date) AND (date_of_birth <= '1984-12-31':date) AND ((country_of_citizenship)::text = 'Philippines':te...))
3	Planning Time: 3.063 ms
4	Execution Time: 0.839 ms

The status bar at the bottom indicates 'Total rows: 4' and 'Query complete 00:00:00.058'. A green message box at the bottom right states: 'Successfully run. Total query runtime: 58 msec. 4 rows affected.'

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

LABWORKS/postgres@PostgreSQL 17

Query

```
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

	indexname	indexdef
1	passengers_pkey	CREATE UNIQUE INDEX passengers_pkey ON public.passengers USING btree (passenger_id)
2	idx_passengers_passport_number	CREATE UNIQUE INDEX idx_passengers_passport_number ON public.passengers USING btree (passport_number)
3	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)

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

Total rows: 3 Query complete 00:00:00.075

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

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

LABWORKS/postgres@PostgreSQL 17

Query

```
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

DROP INDEX

Query returned successfully in 56 msec.

Total rows: Query complete 00:00:00.056

Query returned successfully in 56 msec.