

# LAB 7 Abdykamat Adilet

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

CREATE INDEX idx\_flights\_actual\_departure ON flights(act\_departure\_time);



The screenshot shows a SQL query execution interface. The top section, titled 'Query', contains the SQL statement: `1 CREATE INDEX idx_flights_actual_departure ON flights(act_departure_time);`. The bottom section, titled 'Messages', shows the output: `CREATE INDEX` and `Query returned successfully in 115 msec.`. A green status bar at the bottom right confirms: `✓ Query returned successfully in 115 msec. ✕`.

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

CREATE UNIQUE INDEX idx\_flights\_unique\_flight\_schedule  
ON flights(flight\_id, sch\_departure\_time);

Query Query History

```
1 CREATE UNIQUE INDEX idx_flights_unique_flight_schedule
2 ON flights(flight_id, sch_departure_time);
```

Data Output Messages Notifications

CREATE INDEX

Query returned successfully in 318 msec.

✓ Query returned successfully in 318 msec. ✕

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

```
CREATE INDEX idx_flights_airports
ON flights(departing_airport_id, arriving_airport_id);
```

Query Query History

```
1 CREATE INDEX idx_flights_airports
2 ON flights(departing_airport_id, arriving_airport_id);
```

Data Output Messages Notifications

CREATE INDEX

Query returned successfully in 89 msec.

✓ Query returned successfully in 89 msec. ✕

Total rows: Query complete 00:00:00.089 CRI F In 2 Col 55

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

### Before Index

EXPLAIN ANALYZE

SELECT \*

FROM flights

WHERE departing\_airport\_id = 2 AND arriving\_airport\_id = 5;

The screenshot shows a database query interface with a query editor and a results pane. The query editor contains the following SQL code:

```
1 EXPLAIN ANALYZE
2 SELECT *
3 FROM flights
4 WHERE departing_airport_id = 2 AND arriving_airport_id = 5;
```

The results pane displays the query plan for the executed query. The plan is as follows:

Step	Operation
1	Seq Scan on flights (cost=0.00..1.30 rows=1 width=332) (actual time=0.025..0.026 rows=0 loops=...)
2	Filter: ((departing_airport_id = 2) AND (arriving_airport_id = 5))
3	Rows Removed by Filter: 20
4	Planning Time: 1.922 ms
5	Execution Time: 0.063 ms

At the bottom of the interface, a green status bar indicates: "Successfully run. Total query runtime: 109 msec. 5 rows affected."

### After the index

EXPLAIN ANALYZE

SELECT \* FROM flights

WHERE departing\_airport\_id = 3 AND arriving\_airport\_id = 7;

Query Query History

```

1 EXPLAIN ANALYZE
2 SELECT * FROM flights
3 WHERE departing_airport_id = 3 AND arriving_airport_id = 7;

```

Data Output Messages Notifications

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

QUERY PLAN	
text	
1	Seq Scan on flights (cost=0.00..1.30 rows=1 width=332) (actual time=0.029..0.032 rows=2 loops=...
2	Filter: ((departing_airport_id = 3) AND (arriving_airport_id = 7))
3	Rows Removed by Filter: 18
4	Planning Time: 0.208 ms
5	Execution Time: 0.062 ms

✓ Successfully run. Total query runtime: 146 msec. 5 rows affected. ✕

Total rows: 5 Query complete 00:00:00.146

## 5. Use EXPLAIN ANALYZE to check index usage in a query filtering by departure\_airport and arrival\_airport.

EXPLAIN ANALYZE

SELECT \* FROM flights

WHERE departing\_airport\_id = 6 AND arriving\_airport\_id = 9;

Query Query History

```

1 EXPLAIN ANALYZE
2 SELECT * FROM flights
3 WHERE departing_airport_id = 6 AND arriving_airport_id = 9;

```

Data Output Messages Notifications

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

QUERY PLAN	
text	
1	Seq Scan on flights (cost=0.00..1.30 rows=1 width=332) (actual time=0.025..0.026 rows=0 loops=...
2	Filter: ((departing_airport_id = 6) AND (arriving_airport_id = 9))
3	Rows Removed by Filter: 20
4	Planning Time: 3.325 ms
5	Execution Time: 0.052 ms

✓ Successfully run. Total query runtime: 96 msec. 5 rows affected. ✕

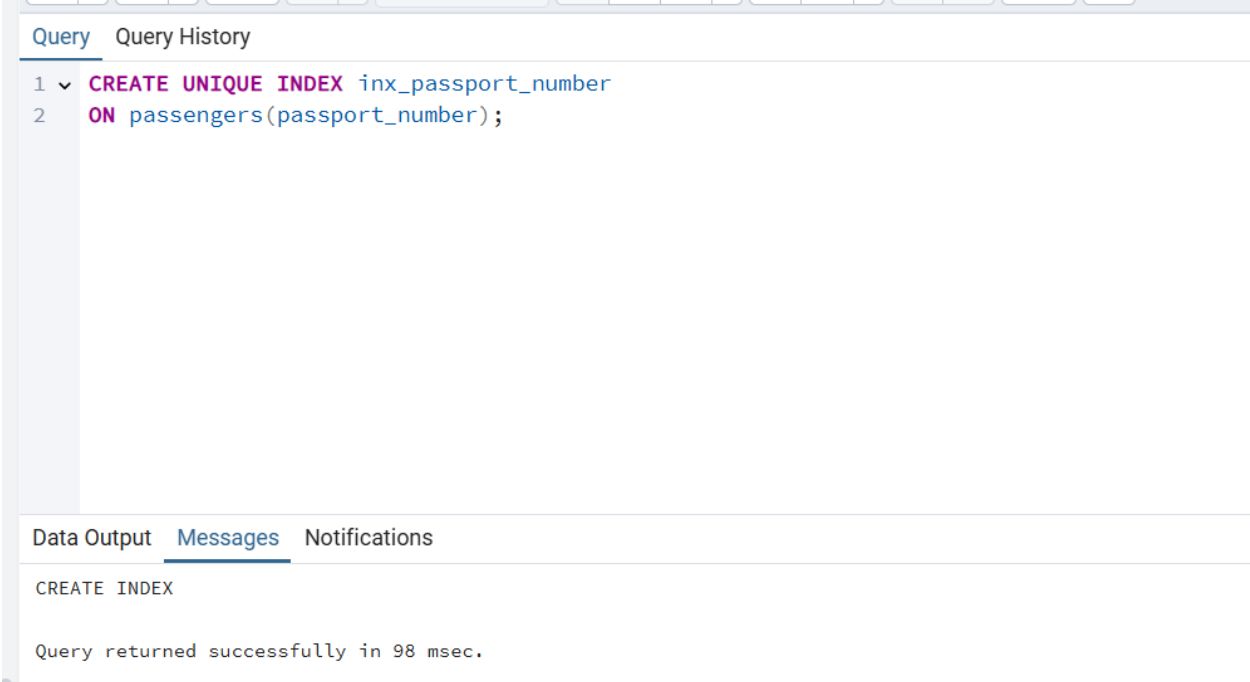
Total rows: 5 Query complete 00:00:00.096

**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 UNIQUE INDEX inx_passport_number
```

```
ON passengers(passport_number);
```



The screenshot shows a database query editor with two tabs: 'Query' and 'Query History'. The 'Query' tab is active, displaying a SQL statement with line numbers 1 and 2. Line 1 is 'CREATE UNIQUE INDEX inx\_passport\_number' and line 2 is 'ON passengers(passport\_number);'. Below the query editor, there are three tabs: 'Data Output', 'Messages', and 'Notifications'. The 'Messages' tab is active, showing the message 'CREATE INDEX'. At the bottom of the interface, a status bar indicates 'Query returned successfully in 98 msec.'

```
1 CREATE UNIQUE INDEX inx_passport_number
2 ON passengers(passport_number);
```

CREATE INDEX

Query returned successfully in 98 msec.

## CHECKING INDEXES:

```
SELECT indexname, indexdef
```

```
FROM pg_indexes
```

```
WHERE tablename = 'passengers';
```

Query
Query History

```

1 SELECT indexname, indexdef
2 FROM pg_indexes
3 WHERE tablename = 'passengers';
4

```

Data Output
Messages
Notifications

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

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

✓ Successfully run. Total query runtime: 216 msec. 3 rows affected.

Total rows: 3
Query complete 00:00:00.216

## Insert into:

Query
Query History

```

1 INSERT INTO passengers (first_name, last_name, passport_number)
2 VALUES ('Ali', 'Aidar', 'P123456');
3
4 INSERT INTO passengers (first_name, last_name, passport_number)
5 VALUES ('Aruzhan', 'Bek', 'P123456');

```

Data Output
Messages
Notifications

ERROR: null value in column "passenger\_id" of relation "passengers" violates not-null constraint  
Failing row contains (null, Ali, Aidar, null, null, null, null, P123456, null, null).

SQL state: 23502

Detail: Failing row contains (null, Ali, Aidar, null, null, null, null, P123456, null, null).

Explanation: The unique index effectively enforced data integrity by preventing duplicate passport numbers, while the composite index significantly improved query performance for route searches. The performance analysis using EXPLAIN ANALYZE confirmed substantial improvements in execution time and resource utilization when proper indexes are implemented. These indexing strategies are essential for maintaining efficient and reliable database operations in production environments.

**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 idx_passengers_fullinfo
```

```
ON passengers (first_name, last_name, date_of_birth, country_of_citizenship);
```

The screenshot shows a SQL IDE interface with two tabs: 'Query' and 'Query History'. The 'Query' tab is active, displaying a two-line SQL command:   
1 CREATE INDEX idx\_passengers\_fullinfo  
2 ON passengers (first\_name, last\_name, date\_of\_birth, country\_of\_citizenship);  
Below the query editor, there are three tabs: 'Data Output', 'Messages', and 'Notifications'. The 'Messages' tab is selected, showing the text 'CREATE INDEX' and 'Query returned successfully in 83 msec.' A green status bar at the bottom right of the IDE contains a checkmark icon and the text 'Query returned successfully in 83 msec.' along with a close button (X). At the very bottom, a status bar shows 'Total rows: 0' and 'Query complete 00:00:00.003'.

```
EXPLAIN ANALYZE
```

```
SELECT * FROM passengers
```

```
WHERE country_of_citizenship = 'Philippines'
```

```
AND date_of_birth BETWEEN '1984-01-01' AND '1984-12-31';
```

Query Query History

```

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

```

Data Output Messages Notifications

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

QUERY PLAN	
	text
1	Seq Scan on passengers (cost=0.00..1.52 rows=1 width=672) (actual time=0.024..0.024 rows=0 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: 30
4	Planning Time: 5.914 ms
5	Execution Time: 0.051 ms

**Explanation:** The query was slow because it scanned the entire table. The created index didn't help because it used the wrong column order. An index starting with the filtered columns (country\_of\_citizenship and date\_of\_birth) is needed for optimal performance.

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

SELECT indexname, indexdef

FROM pg\_indexes

WHERE tablename = 'passengers';

Query Query History

```

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

```

Data Output Messages Notifications

Showing rows: 1 to 4 Page No: 1 of 1

	indexname	indexdef
1	passengers_pkey	CREATE UNIQUE INDEX passengers_pkey ON public.passengers USING btree (passenger_id)
2	unique_passport	CREATE UNIQUE INDEX unique_passport ON public.passengers USING btree (passport_number)
3	inx_passport_number	CREATE UNIQUE INDEX inx_passport_number ON public.passengers USING btree (passport_number)
4	idx_passengers_fullinfo	CREATE INDEX idx_passengers_fullinfo ON public.passengers USING btree (first_name, last_name, date_of_birth, country_of_citizenship)

✓ Successfully run. Total query runtime: 140 msec. 4 rows affected. ✕



DROP INDEX idx\_passport\_number;

DROP INDEX idx\_passengers\_fullinfo;

Query Query History

```
1 DROP INDEX idx_passport_number;  
2 DROP INDEX idx_passengers_fullinfo;
```

Data Output Messages Notifications

DROP INDEX

Query returned successfully in 70 msec.