

**Kazakh-British Technical University**

**DATABASES**  
Laboratory work №7

**KISLITSIN ALEXANDR  
23B031861**

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

The screenshot shows the DataGrip IDE interface. In the Database Explorer, the 'Flights' table is selected. In the console tab, the following SQL command is run:

```
CREATE INDEX idx_actual_departure ON Flights(act_departure_time);
```

The Services panel shows the execution history:

```
[2025-11-11 09:05:59] > CREATE INDEX idx_act_dep ON Flights(act_arrival_time)
[2025-11-11 09:05:59] [30000][1046] No database selected
[2025-11-11 09:06:08] > use airport_db
[2025-11-11 09:06:08] completed in 4 ms
[2025-11-11 09:06:11] airport_db> CREATE INDEX idx_act_dep ON Flights(act_arrival_time)
[2025-11-11 09:06:11] completed in 66 ms
[2025-11-11 09:09:17] airport_db> CREATE INDEX idx_actual_departure ON Flight(act_departure_time)
[2025-11-11 09:09:18] [42S02][1146] Table 'airport_db.flight' doesn't exist
[2025-11-11 09:10:17] airport_db> CREATE INDEX idx_actual_departure ON Flights(act_departure_time)
[2025-11-11 09:10:17] completed in 36 ms
```

At the bottom, the status bar indicates: 486:66 LF UTF-8 4 spaces.

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

The screenshot shows the DataGrip IDE interface. In the Database Explorer, the 'Flights' table is selected. In the console tab, the following SQL command is run:

```
CREATE UNIQUE INDEX unq_idx_flight ON Flights(flight_no, sch_departure_time);
```

The Services panel shows the execution history:

```
[2025-11-11 09:13:07] completed in 49 ms
[2025-11-11 09:13:41] Unsafe query: 'Update' statement without 'where' updates all table rows at once
[2025-11-11 09:13:47] airport_db> UPDATE Flights
      SET flight_no = CONCAT('FL', LPAD(flight_id, 4, '0'))
[2025-11-11 09:13:47] 100 rows affected in 10 ms
[2025-11-11 09:13:59] airport_db> ALTER TABLE Flights
      ADD UNIQUE (Flight_no)
[2025-11-11 09:13:59] completed in 33 ms
[2025-11-11 09:15:49] airport_db> CREATE UNIQUE INDEX unq_idx_flight ON Flights(flight_no, sch_departure_time)
[2025-11-11 09:15:49] completed in 43 ms
```

At the bottom, the status bar indicates: 501:78 LF UTF-8 4 spaces.

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

The screenshot shows the DataGrip IDE interface. In the Database Explorer, under the 'airport\_db' database, the 'tables' node is expanded, showing the 'Flights' table. In the 'console' tab, the following SQL code is run:

```
CREATE UNIQUE INDEX unq_idx_flight ON Flights(flight_no, sch_departure_time);
CREATE INDEX idx_dep_arr_flight ON Flights(departing_airport_id, arriving_airport_id);
SELECT * FROM Flights
WHERE departing_airport_id = 3 AND arriving_airport_id = 10;
```

The output window shows the results of the query:

```
[2025-11-11 20:30:07] Connected
[2025-11-11 20:30:07] airport_db> use airport_db
[2025-11-11 20:30:07] completed in 10 ms
[2025-11-11 20:30:07] airport_db> SELECT * FROM Flights
      WHERE departing_airport_id = 3 AND arriving_airport_id = 10
[2025-11-11 20:30:08] 0 rows retrieved in 410 ms (execution: 49 ms, fetching: 361 ms)
```

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

The screenshot shows the DataGrip IDE interface. In the Database Explorer, under the 'airport\_db' database, the 'tables' node is expanded, showing the 'Flights' table. In the 'console' tab, the following SQL code is run:

```
WHERE departing_airport_id = 3 AND arriving_airport_id = 10;
EXPLAIN ANALYZE
SELECT * FROM Flights
WHERE departing_airport_id = 5 AND arriving_airport_id = 12;
```

The output window shows the results of the query and the execution plan:

```
[2025-11-11 20:30:08] 0 rows retrieved in 410 ms (execution: 49 ms, fetching: 361 ms)
[2025-11-11 20:31:03] airport_db> SELECT * FROM Flights
      WHERE departing_airport_id = 3 AND arriving_airport_id = 10
[2025-11-11 20:31:03] 0 rows retrieved in 383 ms (execution: 19 ms, fetching: 364 ms)
[2025-11-11 20:32:35] airport_db> ALTER TABLE Flights DROP INDEX idx_dep_arr_flight
[2025-11-11 20:32:35] [HY000][1553] Cannot drop index 'idx_dep_arr_flight': needed in a foreign key constraint
[2025-11-11 20:33:29] airport_db> SELECT * FROM Flights
      WHERE departing_airport_id = 5 AND arriving_airport_id = 12
[2025-11-11 20:33:29] 0 rows retrieved in 365 ms (execution: 38 ms, fetching: 335 ms)
```

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 DBeaver interface with the following details:

- Top Bar:** Shows tabs for "DATABASES" and "Version control".
- Left Sidebar:** Database browser showing a tree structure of databases and tables. The "Flights" table under the "localhost" database is selected.
- Central Area:** A SQL editor window titled "console" containing the following code:

```
CREATE UNIQUE INDEX unq_idx_flight ON Flights(flight_no, sch_departure_time);
CREATE INDEX idx_dep_arr_flight ON Flights(departing_airport_id, arriving_airport_id);
EXPLAIN ANALYZE
SELECT * FROM Flights
WHERE departing_airport_id = 5 AND arriving_airport_id = 12;
CREATE UNIQUE INDEX unq_pass_num ON Passengers(passport_number);
```
- Bottom Area:** Services panel and a log window showing the execution of the SQL statements. The log output includes:

```
[2025-11-11 09:15:49] completed in 43 ms
[2025-11-11 09:22:58] airport_db> CREATE INDEX idx_dep_arr_flight ON Flights(departing_airport_id, arriving_airport_id)
[2025-11-11 09:22:58] completed in 47 ms
[2025-11-11 09:24:20] airport_db> SELECT * FROM Flights
[2025-11-11 09:24:20] WHERE departing_airport_id = 5 AND arriving_airport_id = 12
[2025-11-11 09:24:20] 0 rows retrieved in 391 ms (execution: 7 ms, fetching: 384 ms)
[2025-11-11 09:27:16] airport_db> CREATE UNIQUE INDEX unq_pass_num ON Passengers(passport_number)
[2025-11-11 09:27:16] [HY000][1031] Duplicate index 'unq_pass_num' defined on the table 'airport_db.passengers'. This is depr
[2025-11-11 09:27:16] completed in 69 ms
```

The screenshot shows the DataGrip IDE interface. The Database Explorer sidebar on the left lists the schema for the `airport_db` database, including tables like `Airline`, `Flight`, and `Passenger`. The main area is a console window displaying the following SQL code:

```
EXPLAIN ANALYZE
SELECT * FROM Flights
WHERE departing_airport_id = 5 AND arriving_airport_id = 12;

CREATE UNIQUE INDEX unq_pass_num ON Passengers(passport_number);

INSERT INTO Passengers (
    first_name, last_name, date_of_birth, gender,
    country_of_citizenship, country_of_residence, passport_number,
    created_at, updated_at
)
VALUES ('John', 'Doe', '1990-01-01', 'male', 'USA', 'USA', 'A1234567', NOW(), NOW());

CREATE INDEX idx_passenger_name ON Passengers(first_name, last_name, random_date_of_birth);
```

The output pane at the bottom shows the results of the `CREATE INDEX` command, indicating a duplicate key error for the index `unq_pass_num`.

The screenshot shows the DBeaver interface with the following details:

- Top Bar:** Shows "DATABASES" and "Version control".
- Database Explorer:** On the left, under the "airport\_db" schema, the "Passengers" table is selected. The "columns" section shows a column named "passenger\_id" which has a red error icon and the message "[23000][1062] Duplicate entry 'A1234567' for key 'passengers.passport\_number'".
- Console Tab:** The "console" tab is active, showing the SQL code for creating an index and inserting data into the "Passengers" table. The insertion fails at line 529 due to a duplicate entry.
- Services Tab:** Shows a transaction (Tx) and a database connection to "localhost".
- Output Tab:** Shows the execution of the failed insert statement and the resulting error message: "[2025-11-11 20:35:15] 1 row affected in 17 ms [2025-11-11 20:35:51] airport\_db> INSERT INTO Passengers (first\_name, last\_name, date\_of\_birth, gender, country\_of\_citizenship, country\_of\_residence, passport\_number, created\_at, updated\_at) VALUES ('John', 'Doe', '1990-01-01', 'male', 'USA', 'USA', 'A1234567', NOW(), NOW()) [2025-11-11 20:35:51] [23000][1062] Duplicate entry 'A1234567' for key 'passengers.passport\_number'".

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 DataGrip IDE interface. The top navigation bar includes 'DATABASES', 'Version control', and a search bar. The left sidebar, titled 'Database Explorer', shows a tree view of the 'airport\_db' schema, including tables like 'Airline', 'Airport', 'Baggage', etc., and their respective columns and keys. The main area is a 'console' tab with a code editor containing the following SQL queries:

```
532
533
534
535
536
537
538
539
540
541
542 ✓
543
544
545
546
547
548
549
550
551

VALUES ('first_name' 'Mike', 'last_name' 'Smith', 'date_of_birth' '1980-10-01', 'gender' 'male', 'country_of_citizenship' 'USA', 'country_of_residence' 'USA', 'passenger_id' 1);

CREATE INDEX idx_passengers_multi ON Passengers(first_name, last_name, gender, date_of_birth);

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

SHOW INDEXES FROM Passengers;
SHOW INDEXES FROM Flights;
```

The bottom section shows the 'Services' and 'Output' tabs. The 'Output' tab displays the results of the last query, which returned 0 rows. The status bar at the bottom indicates the transaction count (Tx: 1), the current database ('airport\_db'), and the execution time (389 ms).

The screenshot shows the DataGrip IDE interface. In the Database Explorer, the `airport_db` database is selected, and the `Passengers` table is expanded to show columns, keys, and indexes. A query is being run in the `console` tab:

```

CREATE INDEX idx_passengers_multi ON Passengers(first_name, last_name, gender, date_of_birth);
SELECT *
FROM Passengers
WHERE country_of_citizenship = 'Philippines'
AND date_of_birth BETWEEN '1984-01-01' AND '1984-12-31';

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

```

The `Output` tab shows the results of the query execution:

```

[2025-11-11 20:36:47] 0 rows retrieved in 349 ms (execution: 18 ms, fetching: 331 ms)
[2025-11-11 20:37:11] airport_db> SELECT *
    FROM Passengers
    WHERE country_of_citizenship = 'Philippines'
        AND date_of_birth BETWEEN '1984-01-01' AND '1984-12-31'

[2025-11-11 20:37:11] airport_db> SELECT *
    FROM Passengers
    WHERE country_of_citizenship = 'Philippines'
        AND date_of_birth BETWEEN '1984-01-01' AND '1984-12-31'

[2025-11-11 20:37:12] 0 rows retrieved in 347 ms (execution: 22 ms, fetching: 325 ms)

```

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

The screenshot shows the DataGrip IDE interface. In the Database Explorer, the `airport_db` database is selected, and the `Passengers` table is expanded to show columns, keys, and indexes. A query is being run in the `console` tab:

```

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

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

SHOW INDEX FROM Passengers;

```

The `Output` tab shows the results of the query execution, including a table of index details:

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation
Passenger	0	passport_number	1	passport_number	A
Passenger	0	uq_passport	1	passport_number	A
Passenger	0	unq_pass_num	1	passport_number	A
Passenger	1	idx_passengers_multi	1	first_name	A
Passenger	1	idx_passengers_multi	2	last_name	A
Passenger	1	idx_passengers_multi	3	gender	A