

Kazakh British Technical University
Databases

Laboratory work №4

Kislitsin Alexandr
23B031861

Introduction

In this lab, we continued to study SQL queries using the airport database as an example.

The main goal is to learn how to use string functions, conditional expressions, and comparison operators, as well as apply data formatting, filtering, and grouping in queries.

Task 1. Retrieve all airline names in uppercase

The screenshot shows a database console interface with the following components:

- Database Explorer:** Displays the database structure for `lab_db`, including tables `Airline` and `Airport`. The `Airline` table has columns `airline_id`, `airline_code`, `airline_name`, `airline_country`, `created_at`, and `updated_at`.
- Console:** Contains the following SQL queries:

```
253  
254  
255 SELECT *  
256 FROM Airline  
257 ORDER BY created_at DESC  
258 LIMIT 5;  
259  
260  
261  
262  
263 ✓ SELECT UPPER(airline_name) AS airline_name_upper  
264 FROM airline;  
265  
266 SELECT airline_name,  
267 REPLACE(airline_name, 'Air', 'Aero') AS updated_name  
268 FROM airline;  
269  
270  
271 SELECT flight_id, airline_id  
272 FROM flights  
273 WHERE airline_id IN (1, 2);
```
- Output:** Displays the results of the query in line 263, showing the airline names in uppercase:

airline_name_upper
KAZAIR
AIREASY
FLYHIGH
FLYFLY
TURKISHAIR
- Services:** Shows the connection to the database at `localhost`.

Task 2. Replace any occurrence of the word “Air” with “Aero”

The screenshot shows a database management interface with the following components:

- Database Explorer:** Displays the database structure for 'lab_db'. It includes tables 'Airline' and 'Airport'. The 'Airline' table has columns: 'airline_id' (int, auto-increment), 'airline_code' (varchar(30)), 'airline_name' (varchar(50)), 'airline_country' (varchar(50)), 'created_at' (timestamp), and 'updated_at' (timestamp). The 'Airport' table has columns: 'airport_id' (int, auto-increment), 'airport_name' (varchar(50)), 'country' (varchar(50)), and 'state' (varchar(50)).
- Console:** Contains a SQL script with the following queries:

```
253  
254  
255 SELECT *  
256 FROM Airline  
257 ORDER BY created_at DESC  
258 LIMIT 5;  
259  
260  
261  
262  
263 SELECT UPPER(airline_name) AS airline_name_upper  
264 FROM airline;  
265  
266 ✓ SELECT airline_name,  
267      REPLACE(airline_name, 'Air', 'Aero') AS updated_name  
268 FROM airline;  
269  
270  
271 SELECT flight_id, airline_id  
272 FROM flights  
273 WHERE airline_id IN (1, 2);
```
- Services:** Shows the database connection status for '@localhost' and the console execution time (362 ms).
- Output:** Displays the results of the SQL queries. The first query shows the top 5 airlines. The second query shows the updated airline names with 'Air' replaced by 'Aero'. The third query shows the flight details for the first two airlines.

airline_name	updated_name
KazAir	KazAero
AirEasy	AeroEasy
FlyHigh	FlyHigh
FlyFly	FlyFly
TurkishAir	TurkishAero

Database Consoles > @localhost > console 268:14 LF UTF-8 4 spaces

Task 3. Find all flight numbers that coordinate with both airline 1 and airline

The screenshot shows a database IDE interface with the following components:

- Database Explorer:** Displays the database structure for 'lab_db'. It includes tables 'Airline' and 'Airport'. The 'Airline' table has columns: 'airline_id' (int, auto-increment), 'airline_code' (varchar(30)), 'airline_name' (varchar(50)), 'airline_country' (varchar(50)), 'created_at' (timestamp), and 'updated_at' (timestamp). The 'Airport' table has columns: 'airport_id' (int, auto-increment), 'airport_name' (varchar(50)), 'country' (varchar(50)), and 'state' (varchar(50)).
- Console:** Contains the following SQL queries:

```
260  
261  
262  
263 SELECT UPPER(airline_name) AS airline_name_upper  
264 FROM airline;  
265  
266 SELECT airline_name,  
267        REPLACE(airline_name, 'Air', 'Aero') AS updated_name  
268 FROM airline;  
269  
270  
271 ✓ SELECT flight_id, airline_id  
272 FROM flights  
273 WHERE airline_id IN (1, 2);  
274  
275  
276 SELECT *  
277 FROM airport  
278 WHERE airport_name LIKE '%Regional%'  
279 AND airport_name LIKE '%Air%';  
280
```
- Output:** Displays the results of the last query (line 273). It shows a table with 9 rows and 2 columns: 'flight_id' and 'airline_id'.

flight_id	airline_id
1	4
2	6
3	9
4	14
5	19
6	7
7	1
8	1
9	1
- Services:** Shows the 'console' service running on 'localhost'.

Task 4. Retrieve airports that contain “Regional” and “Air” in their names

The screenshot displays a database IDE interface with the following components:

- Database Explorer:** Shows the database structure for `lab_db`. It includes tables `Airline` and `Airport`. The `Airline` table has columns `airline_id` (int, auto-increment), `airline_code` (varchar(30)), `airline_name` (varchar(50)), `airline_country` (varchar(50)), `created_at` (timestamp), and `updated_at` (timestamp). The `Airport` table has columns `airport_id` (int, auto-increment), `airport_name` (varchar(50)), `country` (varchar(50)), and `state` (varchar(50)).
- Console:** Contains SQL queries. The first query is a `REPLACE` statement: `REPLACE(airline_name, 'Air', 'Aero') AS updated_name`. The second query is a `SELECT` statement: `SELECT * FROM airport WHERE airport_name LIKE '%Regional%' AND airport_name LIKE '%Air%';`. The third query is a `SELECT` statement: `SELECT CONCAT(first_name, ' ', last_name) AS full_name, DATE_FORMAT(date_of_birth, '%M %d, %Y') AS formatted_birthdate FROM passengers;`
- Output:** Shows the results of the SQL queries. The first query is highlighted, and the output shows the results of the `REPLACE` operation.
- Services:** Shows the database connection status for `@localhost`.

Task 5. Retrieve passenger names and format their birth dates as ‘Month DD, YYYY’

The screenshot shows a database console interface with the following components:

- Database Explorer:** Displays the database structure for 'lab_db', including tables 'Airline' and 'Airport' with their respective columns and data types.
- Console:** Contains SQL queries. The first query filters for regional airports. The second query joins the 'passengers' table with the 'Airline' table to retrieve passenger names and formatted birth dates.
- Output:** Displays the results of the second query, showing a list of passengers with their full names and birth dates formatted as 'Month DD, YYYY'.

SQL Queries:

```
277 FROM airport
278 WHERE airport_name LIKE '%Regional%'
279 AND airport_name LIKE '%Air%';
280
281
282
283 ✓ SELECT CONCAT(first_name, ' ', last_name) AS full_name,
284       DATE_FORMAT(date_of_birth, '%M %d, %Y') AS formatted_birthdate
285 FROM passengers;
286
287
288 SELECT flight_id,
289        sch_arrival_time,
290        act_arrival_time
291 FROM flights
292 WHERE act_arrival_time > sch_arrival_time;
293
294
295
296 SELECT first_name,
297        last_name,
```

Query Results:

full_name	formatted_birthdate
Fatima Li	December 20, 2004
Algerin Nurpeisova	January 29, 2008
Murat Melnikov	March 19, 1974
Sara Kim	December 28, 1984
Mariya Sokolov	September 24, 2006
Nikita Li	December 30, 1977

Task 6. Find flight numbers that have been delayed

The screenshot shows a database IDE interface with the following components:

- Database Explorer:** Displays the database structure for 'lab_db'. It includes tables 'Airline' and 'Airport'. The 'Airline' table has columns: 'airline_id' (int, auto-increment), 'airline_code' (varchar(30)), 'airline_name' (varchar(50)), 'airline_country' (varchar(50)), 'created_at' (timestamp), and 'updated_at' (timestamp). The 'Airport' table has columns: 'airport_id' (int, auto-increment), 'airport_name' (varchar(50)), 'country' (varchar(50)), and 'state' (varchar(50)).
- Console:** Contains SQL queries. The first query filters 'Airline' records by name patterns:

```
FROM airport
WHERE airport_name LIKE '%Regional%'
AND airport_name LIKE '%Air%';
```

 The second query is a join between 'passengers' and 'flights':

```
SELECT CONCAT(first_name, ' ', last_name) AS full_name,
       DATE_FORMAT(date_of_birth, '%M %d, %Y') AS formatted_birthdate
FROM passengers;
SELECT flight_id,
       sch_arrival_time,
       act_arrival_time
FROM flights
WHERE act_arrival_time > sch_arrival_time;
SELECT first_name,
       last_name,
```
- Output:** Shows the results of the third query, which identifies delayed flights. The columns are 'flight_id', 'sch_arrival_time', and 'act_arrival_time'. The output shows 0 rows.
- Services:** Displays the connection status for '@localhost' and the console execution time (366 ms).

Task 7. Divide passengers into age groups (Young / Adult)

The screenshot shows a database IDE with the following components:

- Database Explorer:** Displays the database structure for 'lab_db', including tables 'Airline' and 'Airport' with their respective columns and data types.
- Console:** Contains two SQL queries. The first query is a SELECT statement that joins 'flights' and 'passengers' tables, calculating the age of passengers based on their date of birth and categorizing them into 'Young' or 'Adult' groups. The second query is a SELECT statement that filters passengers based on their ticket price, categorizing them into 'Cheap' or 'Expensive' groups.
- Output:** Displays the results of the first query, showing a list of passengers with their first name, last name, age, and age group.

SQL Query 1:

```
SELECT first_name, last_name, TIMESTAMPDIFF(YEAR, date_of_birth, CURDATE()) AS age, CASE WHEN TIMESTAMPDIFF(YEAR, date_of_birth, CURDATE()) BETWEEN 18 AND 35 THEN 'Young' WHEN TIMESTAMPDIFF(YEAR, date_of_birth, CURDATE()) BETWEEN 36 AND 55 THEN 'Adult' ELSE 'Other' END AS age_group FROM flights JOIN passengers ON flights.booking_id = passengers.booking_id;
```

SQL Query 2:

```
SELECT booking_id, ticket_price, CASE WHEN ticket_price < 10000 THEN 'Cheap' ELSE 'Expensive' END AS ticket_category FROM passengers;
```

Output Results:

first_name	last_name	age	age_group
Sakura	Yamamoto	25	Young
Fatima	Romanov	25	Young
Daniel	Sokolov	46	Adult
Charlotte	Murpeisova	34	Young
Mei	Murphy	25	Young
Victor	Martin	15	Other

Task 8. Categorize ticket prices as Cheap, Medium or Expensive

The screenshot shows the DBeaver IDE interface. The top panel displays the 'Database Explorer' on the left, showing a tree view of the database structure: 'localhost' > 'lab_db' > 'tables' > 'Airline'. The main panel shows a SQL query in the 'console' tab. The query is a CASE statement that categorizes passengers based on their age and ticket price. The bottom panel shows the 'Output' tab with the results of the query, which is a table with 6 rows and 3 columns: 'booking_id', 'ticket_price', and 'price_category'.

SQL Query:

```

CASE
  WHEN TIMESTAMPDIFF(YEAR, date_of_birth, CURDATE()) BETWEEN 18 AND 35 THEN 'Young'
  WHEN TIMESTAMPDIFF(YEAR, date_of_birth, CURDATE()) BETWEEN 36 AND 55 THEN 'Adult'
  ELSE 'Other'
END AS age_group
FROM passengers;

SELECT booking_id,
       ticket_price,
       CASE
         WHEN ticket_price < 10000 THEN 'Cheap'
         WHEN ticket_price BETWEEN 10000 AND 30000 THEN 'Medium'
         ELSE 'Expensive'
       END AS price_category
FROM booking;

SELECT airline_country,
       COUNT(*) AS airline_count

```

Query Results:

booking_id	ticket_price	price_category
1	17184.00	Medium
2	22180.00	Medium
3	33055.00	Expensive
4	27972.00	Medium
5	33121.00	Expensive
6	42772.00	Expensive

Task 9. Find number of airline names in each country

The screenshot shows a database IDE interface with the following components:

- Database Explorer:** Displays the database structure for 'lab_db'. It includes tables 'Airline' and 'Airport'. The 'Airline' table has columns: 'airline_id' (int, auto increment), 'airline_code' (varchar(30)), 'airline_name' (varchar(50)), 'airline_country' (varchar(50)), 'created_at' (timestamp), and 'updated_at' (timestamp). The 'Airport' table has columns: 'airport_id' (int, auto increment), 'airport_name' (varchar(50)), 'country' (varchar(50)), and 'state' (varchar(50)).
- Console:** Contains two SQL queries. The first query (lines 305-316) selects 'booking_id', 'ticket_price', and a 'price_category' based on 'ticket_price' ranges. The second query (lines 318-322) counts the number of airlines per country, grouped by 'airline_country' and ordered by 'airline_count' in descending order.
- Output:** Displays the results of the second query. It shows a table with two columns: 'airline_country' and 'airline_count'. The results are: Turkey (2), France (1), Brazil (1), and Poland (1).
- Services:** Shows the database connection status for '@localhost'.

```
305
306
307 SELECT booking_id,
308         ticket_price,
309         CASE
310             WHEN ticket_price < 10000 THEN 'Cheap'
311             WHEN ticket_price BETWEEN 10000 AND 30000 THEN 'Medium'
312             ELSE 'Expensive'
313         END AS price_category
314 FROM booking;
315
316
317
318 SELECT airline_country,
319         COUNT(*) AS airline_count
320 FROM airline
321 GROUP BY airline_country
322 ORDER BY airline_count DESC;
323
324
325 SELECT flight_id,
```

airline_country	airline_count
Turkey	2
France	1
Brazil	1
Poland	1

Task 10. Find flights that arrived late

The screenshot shows a database IDE interface with the following components:

- Database Explorer:** Displays the database structure for `lab_db`. It includes tables `Airline` and `Airport`. The `Airline` table has columns `airline_id` (int, auto increment), `airline_code` (varchar(30)), `airline_name` (varchar(50)), `airline_country` (varchar(50)), `created_at` (timestamp), and `updated_at` (timestamp). The `Airport` table has columns `airport_id` (int, auto increment), `airport_name` (varchar(50)), `country` (varchar(50)), and `state` (varchar(50)).
- Console:** Contains two SQL queries. The first query is a grouped count of flights by airline country. The second query is a SELECT statement that identifies late flights by calculating the difference between scheduled and actual arrival times.
- Output:** Shows the results of the second query, which is currently empty (0 rows).

```
314 FROM booking;
315
316
317
318 SELECT airline_country,
319        COUNT(*) AS airline_count
320 FROM airline
321 GROUP BY airline_country
322 ORDER BY airline_count DESC;
323
324
325 SELECT flight_id,
326        sch_arrival_time,
327        act_arrival_time,
328        TIMESTAMPDIFF(MINUTE, sch_arrival_time, act_arrival_time) AS delay_minutes
329 FROM flights
330 WHERE act_arrival_time > sch_arrival_time;
331
332
333
334
```

Output: Result 20

flight_id	sch_arrival_time	act_arrival_time	delay_minutes
-----------	------------------	------------------	---------------

0 rows

Conclusion

During the laboratory work, queries were performed using the string, date, and condition functions.

The skills of formatting texts and dates, using CASE, GROUP BY, LIKE, UPPER, REPLACE, and DATE_FORMAT are fixed.

The result was the ability to create more flexible and informative SQL queries for airport data analysis.