



Kazakh-British Technical University
School of Information Technologies and Engineering

Laboratory work №3
SQL Queries

Prepared by: Gainullin D.
Checked by: Abildayeva T.

Almaty, 2025

CONTENT

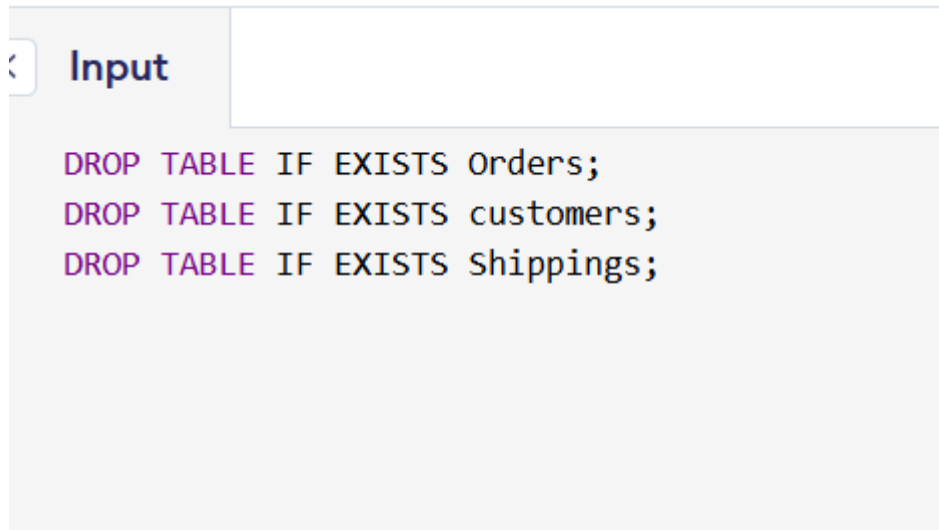
Introduction	3
1. Create the following tables and their relationships:.....	4
2. Fill each table with at least 5 data entries.	6
3. Executing SQL queries.....	9
Conclusion.....	13

INTRODUCTION

In this laboratory work, we learn how to create and manage a relational database using SQL. The main goal of the assignment is to design several related tables — such as `job_positions`, `employees`, `customers`, `products`, `prices`, and `sales_order` — and establish relationships between them using primary and foreign keys. We also practice inserting data, retrieving information with `SELECT` statements, and combining data from multiple tables using `JOIN` and `LEFT JOIN` operations. Through this practical task, students develop skills in basic database design, data manipulation, and SQL query construction.

1. Create the following tables and their relationships:

In this task, we create several relational tables in SQL: Orders, Shippings, Customers,. Each table is defined with appropriate data types, primary keys, and foreign key relationships. This step builds the foundation of the database structure and ensures data integrity between related entities.

A screenshot of a SQL IDE interface. On the left, there is a tab labeled 'Input'. The main area displays three SQL queries, each on a new line: 'DROP TABLE IF EXISTS Orders;', 'DROP TABLE IF EXISTS customers;', and 'DROP TABLE IF EXISTS Shippings;'. The text is color-coded: 'DROP' is purple, 'TABLE' is blue, 'IF' is purple, 'EXISTS' is blue, and the table names are black.

```
DROP TABLE IF EXISTS Orders;
DROP TABLE IF EXISTS customers;
DROP TABLE IF EXISTS Shippings;
```

Figure 1 – Clearing the all data

DROP TABLE IF EXISTS table_name; deletes a table if it already exists. Tables are dropped from child to parent order to avoid foreign key conflicts. This ensures you start with a clean database.

The Output is: SQL query successfully executed. However, the result set is empty.

```
CREATE TABLE job_positions (
  position_id INT PRIMARY KEY,
  position_name VARCHAR(255));
```

```
CREATE TABLE locations (
  location_id INT PRIMARY KEY,
  location_name VARCHAR(255));
```

```
CREATE TABLE customers (
  customer_id INT PRIMARY KEY,
  customer_name VARCHAR(255));
```

```
CREATE TABLE products (
  product_id INT PRIMARY KEY,
  product_name VARCHAR(255));
```

```
CREATE TABLE Departments (
```

```

department_id INT PRIMARY KEY,
department_name VARCHAR(255));

CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    employee_name VARCHAR(255),
    job_title VARCHAR(255));

CREATE TABLE prices (
    product_id INT PRIMARY KEY,
    list_price DECIMAL(10,2),
    FOREIGN KEY (product_id) REFERENCES products(product_id));

CREATE TABLE sales_order (
    order_id INT PRIMARY KEY,
    customer_id INT,
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id));

```

Output: SQL query successfully executed. However, the result set is empty.

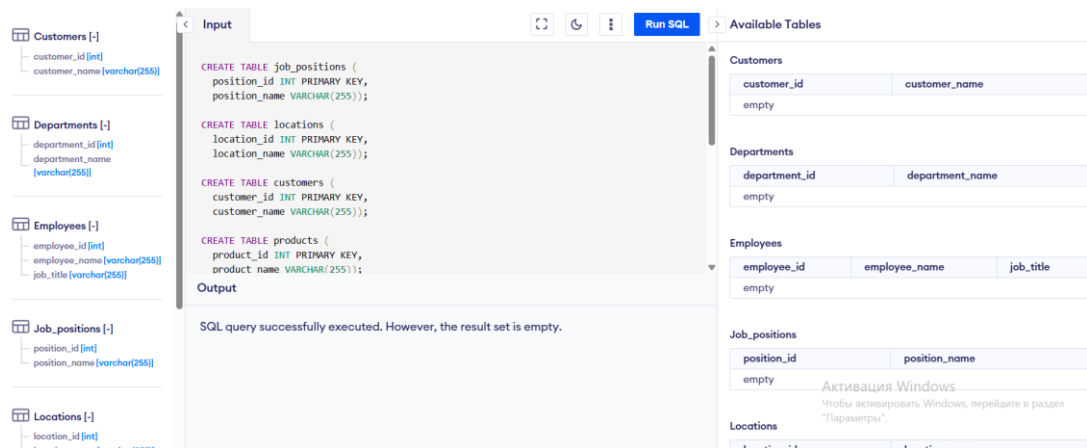


Figure 2 – First part of SQL table creation code

As shown in Figure 2, This figure shows the first part of the SQL script where the tables job_positions, locations, customers, and departments, employees are created. Each table includes primary keys and appropriate data fields to store information about positions, employees, company locations, clients, and products.

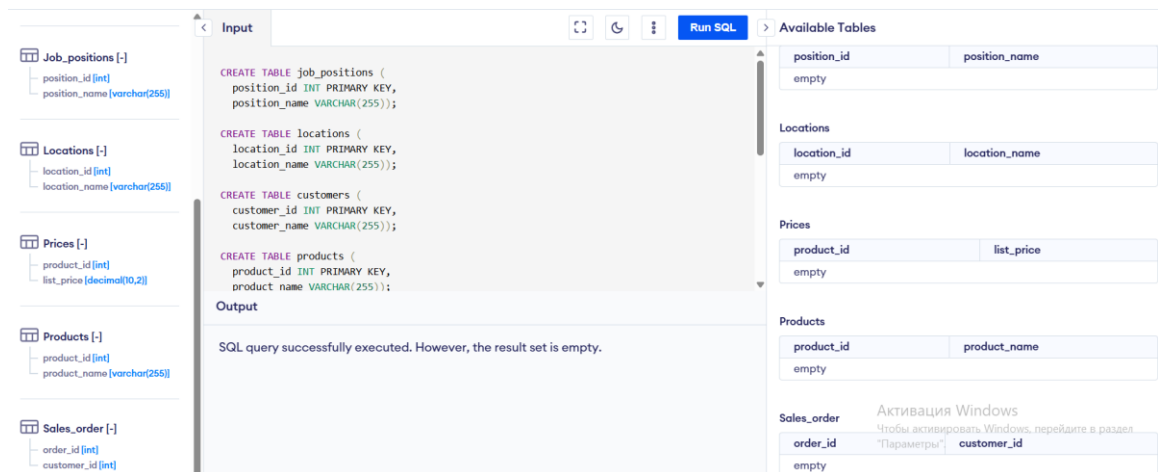


Figure 3 – Second part of SQL table creation code

As presented in Figure 3, This figure presents the second part of the SQL script where the tables prices, products, and sales_orders are created. These tables are linked through primary and foreign keys, forming the complete relational database structure for managing company operations and sales data.

Explanation of columns:

- INT → integer (whole number).
- VARCHAR(255) → text up to 255 characters.
- DECIMAL(10,2) → number with two decimal places (for prices).
- PRIMARY KEY → unique identifier of a row.
- FOREIGN KEY (...) REFERENCES ... → connects this table to another (maintains data consistency).

Each **CREATE TABLE** command creates a new table. **PRIMARY KEY** means the unique identifier of the row. The **FOREIGN KEY** creates a relationship between the tables, for example, the customer_id in sales_order refers to the customer_id in the customers table.

2. Fill each table with at least 5 data entries.

After creating the tables, we insert at least five sample records into each of them using the **INSERT INTO** command. This step fills the database with data that can be later used for testing and executing SQL queries. It helps us understand how to properly input and organize data in a relational database.

```
-- job_positions (5)
INSERT INTO job_positions (position_id, position_name)
VALUES
(1, 'General Manager'),
```

```

(2, 'Drilling Engineer'),
(3, 'Chief Operating Officer'),
(4, 'Reservoir Engineer'),
(5, 'Geologist');

-- locations (5)
INSERT INTO locations (location_id, location_name) VALUES
(1, 'Aktobe'),
(2, 'Almaty'),
(3, 'Astana'),
(4, 'Aktay'),
(5, 'Zharkamys');

-- customers (5)
INSERT INTO customers (customer_id, customer_name) VALUES
(1, 'Kanysh Omirzak'),
(2, 'Beket Adai'),
(3, 'Bakhbergen Kambar'),
(4, 'Dinur Gainullin'),
(5, 'Nurperzent Tilepbayev');

-- sales order (5)
INSERT INTO sales_order (order_id, customer_id) VALUES
(1, 1),
(2, 2),
(3, 3),
(4, 4),
(5, 5);

-- products (5)
INSERT INTO products (product_id, product_name) VALUES
(1, 'Laptop'),
(2, 'Smartphone'),
(3, 'Tablet'),
(4, 'Monitor'),
(5, 'Keyboard');

-- prices (5)
INSERT INTO prices (product_id, list_price) VALUES
(1, 14500.00),
(2, 8000.00),
(3, 9700.00),
(4, 24500.00),
(5, 13200.00);

```

```
-- employees (5)
INSERT INTO employees (employee_id, employee_name,
job_title) VALUES
(1, 'Aman Seitkhan', 'General Manager'),
(2, 'Tapan Masabay', 'Drilling Engineer'),
(3, 'Kulman Ermaganbet', 'Chief Operating Officer'),
(4, 'Tilepbay Masabay', 'Reservoir Engineer'),
(5, 'Aron Osken', 'Geologist');

-- departments (5)
INSERT INTO Departments (department_id, department_name)
VALUES
(1, 'Sales'),
(2, 'Drilling'),
(3, 'Operation'),
(4, 'Engineer'),
(5, 'Geology');
```

Available Tables

Customers

customer_id	customer_name
1	Kanysh Omirzak
2	Beket Adai
3	Bakhbergen Kambar
4	Dinur Gainullin
5	Nurperzent Tilepbayev

Departments

department_id	department_name
1	Sales
2	Drilling
3	Operation
4	Engineer
5	Geology

Employees

employee_id	employee_name	job_title
1	Aman Seitkhan	General Manager
2	Tapan Masabay	Drilling Engineer
3	Kulman Ermaganbet	Chief Operating Officer
4	Tilepbay Masabay	Reservoir Engineer
5	Aron Osken	Geologist

Job_positions

position_id	position_name
1	General Manager
2	Drilling Engineer
3	Chief Operating Officer
4	Reservoir Engineer
5	Geologist

Locations

location_id	location_name
1	Almaty
2	Almaty
3	Almaty
4	Almaty
5	Zharkamys

Figure 3 – First part of SQL data insertion code

This figure shows the first part of the SQL INSERT INTO statements used to add data into the tables job_positions, locations, customers, and departments, employees. Each table was filled with at least five records to provide sample data for further SQL queries and testing.

Prices	
product_id	list_price
1	14500
2	8000
3	9700
4	24500
5	13200

Products	
product_id	product_name
1	Laptop
2	Smartphone
3	Tablet
4	Monitor
5	Keyboard

Sales_order	
order_id	customer_id
1	1
2	2
3	3
4	4
5	5

Figure 4 – Second part of SQL data insertion code

This figure presents the second part of the SQL data insertion, where records were added into the tables prices, products, and sales_orders tables. These data entries establish relationships between products, prices, customers, and orders, allowing the database to demonstrate how relational links work in practice.

Explanation: Each **INSERT INTO ... VALUES (...)** command adds a new row into the specified table. Foreign keys (like customer_id in sales_order) must already exist in the parent table (customers).

3. Executing SQL queries

In these tasks, we perform different SQL queries to retrieve and display specific information from the database. First, we use a simple **SELECT * FROM job_positions;** command to show all job positions stored in the table. Then, we execute **SELECT * FROM locations ORDER BY location_id DESC;** to display all locations sorted in descending order by their IDs. Next, we apply an **INNER JOIN** between the sales_order and customers tables to show each order with the corresponding customer name. Finally, we use a **LEFT JOIN** between the products and prices tables to display all products together with their prices, including those that

do not have a price assigned (which appear as NULL). These queries help us understand how to retrieve, sort, and combine data from multiple tables effectively.

1. Show all job positions: **SELECT * FROM** job_positions;

position_id	position_name
1	General Manager
2	Drilling Engineer
3	Chief Operating Officer
4	Reservoir Engineer
5	Geologist

Figure 5 – All positions

2. Show all locations in descending order by ID:

```
SELECT * FROM locations  
ORDER BY location_id DESC;
```

location_id	location_name
5	Zharkamys
4	Aktay
3	Astana
2	Almaty
1	Aktobe

Figure 6 – Sorted by location

3. Show all orders with customer names:

```
SELECT s.order_id, c.customer_name  
FROM sales_order s  
JOIN customers c ON s.customer_id = c.customer_id;
```

order_id	customer_name
1	Kanysh Omirzak
2	Beket Adai
3	Bakhbergen Kambar
4	Dinur Gainullin
5	Nurperzent Tilepbayev

Figure 7 - JOIN customers

4. Show all products and their prices (including those without price):

```
SELECT p.product_name, pr.list_price
FROM products p
LEFT JOIN prices pr ON p.product_id = pr.product_id;
```

product_name	list_price
Laptop	14500
Smartphone	8000
Tablet	9700
Monitor	24500
Keyboard	13200

Figure 8 – List price

5. SQL query for getting all orders with customer names (JOIN).

```
SELECT so.order_id, c.customer_name
FROM sales_order so
JOIN customers c ON so.customer_id = c.customer_id;
```

order_id	customer_name
1	Kanysh Omirzak
2	Beket Adai
3	Bakhbergen Kambar
4	Dinur Gainullin
5	Nurperzent Tilepbayev

Figure 9 – Customers

6. SQL query for getting all products with their prices (LEFT JOIN).

```
SELECT p.product_name, pr.list_price
FROM products p
LEFT JOIN prices pr ON p.product_id = pr.product_id;
```

product_name	list_price
Laptop	14500
Smartphone	8000
Tablet	9700
Monitor	24500
Keyboard	13200

Figure 10 – Prices

Explanation of SQL keywords:

- **SELECT** — choose which columns to display.
- **FROM** — specifies the table.
- **JOIN** — returns only matching rows between two tables.
- **LEFT JOIN** — returns all rows from the left table and matching rows from the right table; missing matches show NULL.
- **ORDER BY ... DESC** — sorts data in descending order.

```
SELECT *  
FROM job_positions;  
SELECT *  
FROM locations ORDER BY location_id DESC;  
SELECT so.order_id, c.customer_name  
FROM sales_order so  
JOIN customers c ON so.customer_id = c.customer_id;  
SELECT p.product_name, pr.list_price  
FROM products p  
LEFT JOIN prices pr ON p.product_id = pr.product_id;  
SELECT so.order_id, c.customer_name  
FROM sales_order so  
JOIN customers c ON so.customer_id = c.customer_id;  
SELECT p.product_name, pr.list_price  
FROM products p  
LEFT JOIN prices pr ON p.product_id = pr.product_id;
```

CONCLUSION

As a result of this laboratory work, we successfully created and populated multiple database tables, connected them using relational keys, and performed various SQL queries to extract and analyze data. This exercise helped to better understand how databases are structured and how different tables can interact through relationships. The practical experience gained will be useful for future work in database management and application development, where efficient data organization and retrieval are essential.