НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування та спеціалізованих комп'ютерних систем

Розрахунково-графічна робота

з дисципліни **Бази даних і засоби управління**

на тему: " Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL"

Виконав: студент III курсу

ФПМ групи КВ-23

Атанов Назар

Перевірив:

Київ – 2024

https://github.com/NazarBello/BD_lab.git

Метою роботи є здобуття вмінь програмування прикладних додатків баз даних PostgreSQL.

Завдання роботи полягає у наступному:

1. Реалізувати функції внесення, редагування та вилучення даних у таблицях бази даних, створених у лабораторній роботі №1, засобами консольного інтерфейсу.
2. Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі.
3. Забезпечити реалізацію пошуку за декількома атрибутами з двох та більше сутностей одночасно: для числових атрибутів – у рамках діапазону, для рядкових – як шаблон функції LIKE оператора SELECT SQL, для логічного типу – значення True/False, для дат – у рамках діапазону дат.
4. Програмний код виконати згідно шаблону MVC (модель-подання-контролер).

Виконання роботи

Логічна модель предметної області «Система бронювань подорожей»

Логічну модель (схему бази даних) наведено на рисунку 1.

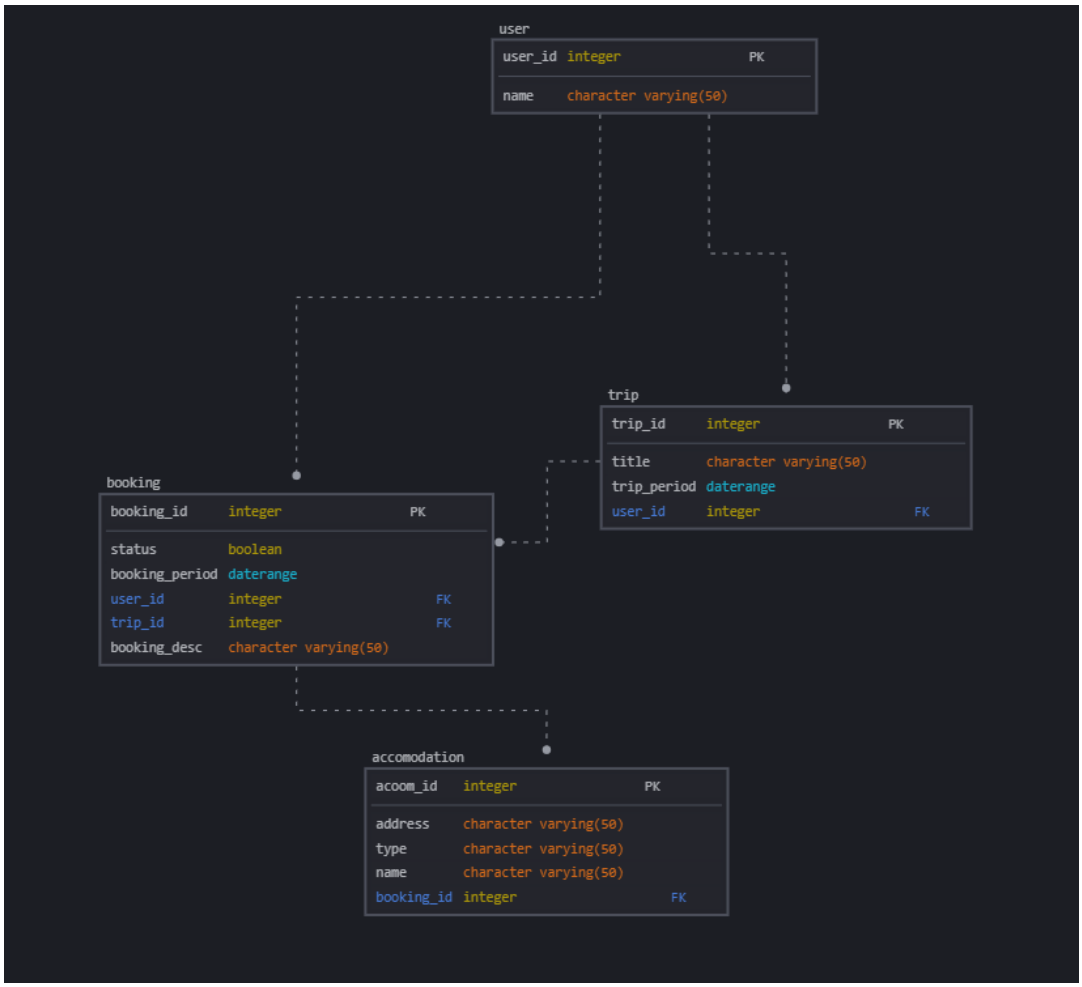

Рисунок 1 -Логічна модель бази даних

Зміни у порівнянні з першою лабораторною роботою відсутні. (інструмент: sqldbm.com)

Середовище та компоненти розробки

Для розробки використовувалась мова програмування Java, середовище розробки Intellij Idea Ultimate, сторонній набір бібліотек JDBC також використаний Maven для dependencies

Шаблон проектування

MVC - шаблон проектування, який використаний у програмі.

Model – представляє клас, що описує логіку використовуваних даних. Згідно компоненту моделі, у моїй програмі відповідають всі компоненти які знаходяться у папці Models.

View – в нашому випадку консольний інтерфейс з яким буде взаємодіяти наш користувач. Згідно компоненту представлення, то їй відповідають такі компоненти, згідно яким користувач бачить необхідні дані, що є представленням даних у вигляді консольного інтерфейсу.

Controller – представляє клас, що забезпечує зв'язок між користувачем і системою, поданням і сховищем даних. Він отримує вводяться користувачем дані і обробляє їх. І в залежності від результатів обробки відправляє користувачеві певний висновок, наприклад, у вигляді подання.

Структура програми та її опис

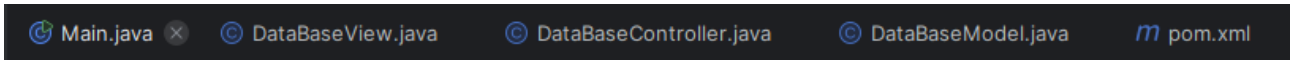

Рисунок 2 - Структура програми

Програма умовно розділена на 4 модулі: файл DataBaseController.java, файл DataBaseModel.java та файл DataBaseView.java та головний файл Main.java

Класи, як видно з їх назв, повністю відповідають використаному патерну MVC.

У файлі DataBaseModel описаний клас моделі, що займається регулюванням підключення до бази даних, та виконанням запитів до неї.

У файлі DataBaseController описаний інтерфейс взаємодії з користувачем, запит бажаної дії, виконання пошуку, тощо.

У файлі DataBaseView описаний клас, що виводить результати виконання тієї чи іншої дії на екран консолі.

Структура меню програми

```
Choose what action you would like to perform
1. Insert data into the database
2. Remove data from the database
3. Edit data in the database
4. Generate random data into the database
5. Search data in the database
6. Print data from the database
```

Рисунок 3 - Меню програми

Меню користувача складається з шести пунктів (Рисунок 3).

Перший пункт пропонує введення даних в таблицю

Другий пункт пропонує видалення даних з таблиці

Третій пункт пропонує оновлення даних в таблиці

Четвертий пункт пропонує генерування даних в таблиці

П’ятий пункт пропонує пошук даних у таблиці

Шостий пункт подає таблиці в текстовому представленні

Insert Delete Update

Insert operation

	booking_id [PK] integer	users_id integer	trip_id integer	status boolean	booking_period daterange	booking_desc character varying (1000)
1	1	1	1	true	[2024-01-06,2025-06-11)	one room flat
2	2	7	2	true	[2024-01-12,2024-08-14)	castle
3	3	3	3	false	[2024-01-20,2025-12-25)	townhouse

```
Choose a table to insert data:
0: booking
1: accomodation
2: users
3: trip
Choose a table to insert data:
0
Enter value for -> booking_id (integer):
4
Enter value for -> users_id (integer):
3
Enter value for -> trip_id (integer):
2
Enter value for -> status (boolean):
false
Enter value for -> booking_period (daterange):
[2024-09-10,2025-12-15]
Enter value for -> booking_desc (character varying):
hotel room
Data inserted successfully into booking
```

	booking_id [PK] integer	users_id integer	trip_id integer	status boolean	booking_period daterange	booking_desc character varying (1000)
1	1	1	1	true	[2024-01-06,2025-06-11)	one room flat
2	2	7	2	true	[2024-01-12,2024-08-14)	castle
3	3	3	3	false	[2024-01-20,2025-12-25)	townhouse
4	4	3	2	false	[2024-09-10,2025-12-16)	hotel room

```
Choose a table to delete data:
0: booking
1: accomodation
2: users
3: trip
Choose a table to delete data:
2
Available columns for filtering:
0: users_id (integer)
1: name (character varying)
Choose a column to filter rows for deletion:
0
Enter the value for users_id:
3
1 row(s) deleted from users
```

Після видалення юзера з id = 3, через наявність каскаду видаляються і елементи з таблиці booking з тим самим user_id

	booking_id [PK] integer	users_id integer	trip_id integer	status boolean	booking_period daterange	booking_desc character varying (1000)
1	1	1	1	true	[2024-01-06,2025-06-11)	one room flat
2	2	7	2	true	[2024-01-12,2024-08-14)	castle

Generate

```
Choose what action you would like to perform
1. Insert data into the database
2. Remove data from the database
3. Edit data in the database
4. Generate random data into the database
5. Search data in the database
6. Print data from the database
4
Choose a table to generate data:
0: booking
1: accomodation
2: users
3: trip
Choose a table to generate data:
2
Enter the number of records to generate:
100000
100000 random records inserted into users
Query execution time consumption: 250
```

Копія запиту

```
"INSERT INTO users (name) " +
        "SELECT 'User' || trunc(random() * 100)::int FROM generate_series(1, " +
count + ")";
```

Скріншот з бази данних

2231	2231	User75
2232	2232	User26
2233	2233	User58
2234	2234	User86
2235	2235	User49
2236	2236	User52
2237	2237	User19
2238	2238	User12
2239	2239	User15
2240	2240	User16
2241	2241	User43

```
Choose what action you would like to perform
1. Insert data into the database
2. Remove data from the database
3. Edit data in the database
4. Generate random data into the database
5. Search data in the database
6. Print data from the database
4
Choose a table to generate data:
0: booking
1: accomodation
2: users
3: trip
Choose a table to generate data:
0
Enter the number of records to generate:
15
15 random records inserted into booking
Query execution time consumption: 8
```

1	1	1	1	true	[2024-01-06,2025-06-11)	one room flat
2	2	7	2	true	[2024-01-12,2024-08-14)	castle
3	70	6490	2	false	[2024-02-16,2025-04-01)	BookingDesc_46
4	71	6490	2	false	[2024-02-16,2025-04-01)	BookingDesc_69
5	72	6490	2	true	[2024-02-16,2025-04-01)	BookingDesc_7
6	73	6490	2	true	[2024-02-16,2025-04-01)	BookingDesc_66
7	74	6490	2	true	[2024-02-16,2025-04-01)	BookingDesc_68
8	75	6490	2	false	[2024-02-16,2025-04-01)	BookingDesc_96
9	76	6490	2	true	[2024-02-16,2025-04-01)	BookingDesc_88
10	77	6490	2	false	[2024-02-16,2025-04-01)	BookingDesc_89
11	78	6490	2	true	[2024-02-16,2025-04-01)	BookingDesc_38
12	79	6490	2	true	[2024-02-16,2025-04-01)	BookingDesc_89
13	80	6490	2	false	[2024-02-16,2025-04-01)	BookingDesc_29
14	81	6490	2	false	[2024-02-16,2025-04-01)	BookingDesc_62
15	82	6490	2	true	[2024-02-16,2025-04-01)	BookingDesc_39
16	83	6490	2	false	[2024-02-16,2025-04-01)	BookingDesc_96
17	84	6490	2	false	[2024-02-16,2025-04-01)	BookingDesc_3

Копія SQL запиту

```
"INSERT INTO booking (users_id, trip_id, status, booking_period, book-
ing_desc)\n" +
        "SELECT\n" +
        "    u.users_id,\n" +
        "    tr.trip_id,\n" +
        "    (random() > 0.5) AS status,\n" +
        "    daterange(\n" +
        "        LEAST(date '2024-01-01' + d.day1, date '2024-01-01' +
```

```
d.day2),\n" +
        "        GREATEST(date '2024-01-01' + d.day1, date '2024-01-01' +
d.day2)\n" +
        "    ) AS booking_period,\n" +
        "    'BookingDesc_' || trunc(random() * 100)::int AS booking_desc\n" +
        "FROM generate_series(1, " + count + ") g\n" +
        "CROSS JOIN LATERAL (\n" +
        "    -- Approximately 5 years * 365 ~ 1825 days\n" +
        "    SELECT trunc(random() * 1825)::int AS day1,\n" +
        "           trunc(random() * 1825)::int AS day2\n" +
        ") d\n" +
        "CROSS JOIN LATERAL (\n" +
        "    SELECT users_id\n" +
        "    FROM users\n" +
        "    OFFSET floor(random() * (SELECT count(*) FROM users))\n" +
        "    LIMIT 1\n" +
        ") u\n" +
        "CROSS JOIN LATERAL (\n" +
        "    SELECT trip_id\n" +
        "    FROM trip\n" +
        "    OFFSET floor(random() * (SELECT count(*) FROM trip))\n" +
        "    LIMIT 1\n" +
        ") tr;";
```

Search

1	1	1	1	true	[2024-01-06,2025-06-11)	one room flat
2	2	7	2	true	[2024-01-12,2024-08-14)	castle
3	70	6490	2	false	[2024-02-16,2025-04-01)	BookingDesc_46
4	71	6490	2	false	[2024-02-16,2025-04-01)	BookingDesc_69
5	72	6490	2	true	[2024-02-16,2025-04-01)	BookingDesc_7
6	73	6490	2	true	[2024-02-16,2025-04-01)	BookingDesc_66
7	74	6490	2	true	[2024-02-16,2025-04-01)	BookingDesc_68
8	75	6490	2	false	[2024-02-16,2025-04-01)	BookingDesc_96
9	76	6490	2	true	[2024-02-16,2025-04-01)	BookingDesc_88
10	77	6490	2	false	[2024-02-16,2025-04-01)	BookingDesc_89
11	78	6490	2	true	[2024-02-16,2025-04-01)	BookingDesc_38
12	79	6490	2	true	[2024-02-16,2025-04-01)	BookingDesc_89
13	80	6490	2	false	[2024-02-16,2025-04-01)	BookingDesc_29
14	81	6490	2	false	[2024-02-16,2025-04-01)	BookingDesc_62
15	82	6490	2	true	[2024-02-16,2025-04-01)	BookingDesc_39
16	83	6490	2	false	[2024-02-16,2025-04-01)	BookingDesc_96
17	84	6490	2	false	[2024-02-16,2025-04-01)	BookingDesc_3

```
Choose tables to search data (comma-separated indices):
0
Columns in table: booking
 - booking_id
 - users_id
 - trip_id
 - status
 - booking_period
 - booking_desc
Enter filtering conditions (e.g., table1.column1=value AND table2.column2=value):
status = true
Enter columns to group by (comma-separated, e.g., table1.column1, table2.column2):
booking_id
Query Results:
booking_id: 1 | users_id: 1 | trip_id: 1 | status: t | booking_period: [2024-01-06,2025-06-11) | booking_desc: one room flat  |
booking_id: 2 | users_id: 7 | trip_id: 2 | status: t | booking_period: [2024-01-12,2024-08-14) | booking_desc: castle |
booking_id: 72 | users_id: 6490 | trip_id: 2 | status: t | booking_period: [2024-02-16,2025-04-01) | booking_desc: BookingDesc_7 |
booking_id: 73 | users_id: 6490 | trip_id: 2 | status: t | booking_period: [2024-02-16,2025-04-01) | booking_desc: BookingDesc_66 |
booking_id: 74 | users_id: 6490 | trip_id: 2 | status: t | booking_period: [2024-02-16,2025-04-01) | booking_desc: BookingDesc_68 |
booking_id: 76 | users_id: 6490 | trip_id: 2 | status: t | booking_period: [2024-02-16,2025-04-01) | booking_desc: BookingDesc_88 |
booking_id: 78 | users_id: 6490 | trip_id: 2 | status: t | booking_period: [2024-02-16,2025-04-01) | booking_desc: BookingDesc_38 |
booking_id: 79 | users_id: 6490 | trip_id: 2 | status: t | booking_period: [2024-02-16,2025-04-01) | booking_desc: BookingDesc_89 |
booking_id: 82 | users_id: 6490 | trip_id: 2 | status: t | booking_period: [2024-02-16,2025-04-01) | booking_desc: BookingDesc_39 |
Query executed in 4 ms.
```

```
Columns in table: booking
 - booking_id
 - users_id
 - trip_id
 - status
 - booking_period
 - booking_desc
Enter filtering conditions (e.g., table1.column1=value AND table2.column2=value):
booking_id = 74 AND users_id = 6490 AND status = true
Enter columns to group by (comma-separated, e.g., table1.column1, table2.column2):
booking_id
Query Results:
booking_id: 74 | users_id: 6490 | trip_id: 2 | status: t | booking_period: [2024-02-16,2025-04-01) | booking_desc: BookingDesc_68 |
Query executed in 1 ms.
```

Опис модуля «Model»

Програмний код модуля

```java
package org.example;

import java.sql.*;
import java.util.*;
import java.util.regex.Pattern;

public class DataBaseModel {
    private final Connection connection;

    public DataBaseModel(String jdbcUrl) throws SQLException {
        connection = DriverManager.getConnection(jdbcUrl);
        System.out.println("Connection established");
    }

    /**
     * Insert data into a selected table with validation for data types and for-
eign keys.
     */
    public void insert() throws SQLException {
        Scanner scanner = new Scanner(System.in);
        List<String> tables = getTables("insert");

        System.out.println("Choose a table to insert data:");
        int tableIndex = scanner.nextInt();
```

```
        scanner.nextLine(); // Consume newline

        String selectedTable = tables.get(tableIndex);
        List<String> columns = getColumns(selectedTable);
        List<String> columnTypes = getColumnTypes(selectedTable);

        List<String> values = new ArrayList<>();

        for (int i = 0; i < columns.size(); i++) {
            String column = columns.get(i);
            String columnType = columnTypes.get(i);
            System.out.println("Enter value for -> " + column + " (" + column-
Type + "):");

            String value = scanner.nextLine();

            if (isForeignKey(column, selectedTable)) {
                // Check if the foreign key value exists in the parent table
                if (!validateForeignKey(column, Integer.parseInt(value))) {
                    System.out.println("Invalid foreign key value for column: "
+ column);
                    return;
                }
            }

            if (columnType.equals("character varying")) {
                value = "'" + value.replace("'", "''") + "'";
            } else if (columnType.equals("daterange")) {
                // Validate and format daterange
                if (!validateDateRange(value)) {
                    System.out.println("Invalid date range format. Use [YYYY-MM-
DD,YYYY-MM-DD].");
                    return;
                }
                value = "'" + value + "'";
            }

            values.add(value);
        }

        String query = "INSERT INTO " + selectedTable + " (" + String.join(", ",
columns) + ") VALUES (" + String.join(", ", values) + ")";
        executeUpdate(query, "Data inserted successfully into " + select-
edTable);
    }

    /**
     * Generate random data using SQL queries for selected table.
     */
    public void generate() throws SQLException {
        Scanner scanner = new Scanner(System.in);
        List<String> tables = getTables("generate");

        System.out.println("Choose a table to generate data:");
        int tableIndex = scanner.nextInt();
        scanner.nextLine(); // Consume newline

        System.out.println("Enter the number of records to generate:");
        int recordCount = scanner.nextInt();
        long time_start = System.nanoTime();
        String selectedTable = tables.get(tableIndex);
        String query = generateSQLForGeneration(selectedTable, recordCount);

        executeUpdate(query, recordCount + " random records inserted into " +
selectedTable);
        long time_end = System.nanoTime();
```

```
        long result = (time_end - time_start)/ 1_000_000;
        System.out.println("Query execution time consumption: " + result);

    }


    public void search() throws SQLException {
        Scanner scanner = new Scanner(System.in);
        List<String> tables = getTables("search");

        // Step 1: Select tables
        System.out.println("Choose tables to search data (comma-separated indi-
ces):");
        String[] selectedIndices = scanner.nextLine().split(",");
        List<String> selectedTables = new ArrayList<>();
        for (String index : selectedIndices) {
            selectedTables.add(tables.get(Integer.parseInt(index.trim())));
        }

        // Step 2: Display columns for each selected table
        Map<String, List<String>> tableColumns = new HashMap<>();
        for (String table : selectedTables) {
            List<String> columns = getColumns(table);
            tableColumns.put(table, columns);
            System.out.println("Columns in table: " + table);
            for (String column : columns) {
                System.out.println(" - " + column);
            }
        }

        // Step 3: Input filtering conditions
        System.out.println("Enter filtering conditions (e.g., table1.col-
umn1=value AND table2.column2=value):");
        String filters = scanner.nextLine();

        // Step 4: Input grouping columns
        System.out.println("Enter columns to group by (comma-separated, e.g.,
table1.column1, table2.column2):");
        String groupBy = scanner.nextLine();

        // Step 5: Construct the query
        StringBuilder query = new StringBuilder("SELECT ");
        for (String table : selectedTables) {
            for (String column : tableColumns.get(table)) {
                query.append(table).append(".").append(column).append(", ");
            }
        }
        query.setLength(query.length() - 2); // Remove the last comma and space
        query.append(" FROM ").append(String.join(", ", selectedTables));
        if (!filters.isEmpty()) {
            query.append(" WHERE ").append(filters);
        }
        if (!groupBy.isEmpty()) {
            query.append(" GROUP BY ").append(groupBy);
        }

        // Step 6: Measure query execution time
        long startTime = System.currentTimeMillis();
        try (PreparedStatement preparedStatement = connection.prepareState-
ment(query.toString());
             ResultSet resultSet = preparedStatement.executeQuery()) {

            // Step 7: Display results
            ResultSetMetaData metaData = resultSet.getMetaData();
            int columnCount = metaData.getColumnCount();
```

```
            System.out.println("Query Results:");
            while (resultSet.next()) {
                for (int i = 1; i <= columnCount; i++) {
                    System.out.print(metaData.getColumnName(i) + ": " + re-
sultSet.getString(i) + " | ");
                }
                System.out.println();
            }
        }
        long endTime = System.currentTimeMillis();

        // Step 8: Output execution time
        System.out.println("Query executed in " + (endTime - startTime) + "
ms.");
    }


    public void delete() throws SQLException {
        Scanner scanner = new Scanner(System.in);
        List<String> tables = getTables("delete");

        System.out.println("Choose a table to delete data:");
        int tableIndex = scanner.nextInt();
        scanner.nextLine(); // Consume newline

        String selectedTable = tables.get(tableIndex);
        List<String> columns = getColumns(selectedTable);
        List<String> columnTypes = getColumnTypes(selectedTable);

        System.out.println("Available columns for filtering:");
        for (int i = 0; i < columns.size(); i++) {
            System.out.println(i + ": " + columns.get(i) + " (" + column-
Types.get(i) + ")");
        }

        System.out.println("Choose a column to filter rows for deletion:");
        int columnIndex = scanner.nextInt();
        scanner.nextLine(); // Consume newline

        String selectedColumn = columns.get(columnIndex);
        String columnType = columnTypes.get(columnIndex);

        System.out.println("Enter the value for " + selectedColumn + ":");
        String filterValue = scanner.nextLine();

        String query = "DELETE FROM " + selectedTable + " WHERE " + selectedCol-
umn + " = ?";
        try (PreparedStatement preparedStatement = connection.prepareState-
ment(query)) {
            // Handle type-specific parsing
            if (columnType.equals("integer")) {
                preparedStatement.setInt(1, Integer.parseInt(filterValue));
            } else if (columnType.equals("boolean")) {
                preparedStatement.setBoolean(1, Boolean.parseBoolean(filter-
Value));
            } else if (columnType.equals("daterange")) {
                preparedStatement.setObject(1, filterValue,
java.sql.Types.OTHER); // Daterange in PostgreSQL
            } else {
                preparedStatement.setString(1, filterValue); // Default for text
or character types
            }

            int rowsAffected = preparedStatement.executeUpdate();

            if (rowsAffected > 0) {
```

```
                System.out.println(rowsAffected + " row(s) deleted from " + se-
lectedTable);
            } else {
                System.out.println("No rows matched the filter.");
            }
        } catch (SQLException e) {
            System.err.println("Error executing delete: " + e.getMessage());
            throw e;
        } catch (NumberFormatException e) {
            System.err.println("Invalid input type: " + e.getMessage());
        }
    }



    public void update() throws SQLException {
        Scanner scanner = new Scanner(System.in);
        List<String> tables = getTables("update");

        System.out.println("Choose a table to update data:");
        int tableIndex = scanner.nextInt();
        scanner.nextLine(); // Consume newline

        String selectedTable = tables.get(tableIndex);
        List<String> columns = getColumns(selectedTable);

        System.out.println("Choose a column to filter rows for updating:");
        for (int i = 0; i < columns.size(); i++) {
            System.out.println(i + ": " + columns.get(i));
        }

        int columnIndex = scanner.nextInt();
        scanner.nextLine(); // Consume newline

        String filterColumn = columns.get(columnIndex);
        System.out.println("Enter the value for " + filterColumn + ":");
        String filterValue = scanner.nextLine();

        System.out.println("Enter the column to update:");
        int updateColumnIndex = scanner.nextInt();
        scanner.nextLine(); // Consume newline

        String updateColumn = columns.get(updateColumnIndex);
        System.out.println("Enter the new value for " + updateColumn + ":");
        String newValue = scanner.nextLine();

        String query = "UPDATE " + selectedTable + " SET " + updateColumn + " =
? WHERE " + filterColumn + " = ?";
        try (PreparedStatement preparedStatement = connection.prepareState-
ment(query)) {
            preparedStatement.setString(1, newValue);
            preparedStatement.setString(2, filterValue);
            int rowsAffected = preparedStatement.executeUpdate();

            if (rowsAffected > 0) {
                System.out.println(rowsAffected + " row(s) updated in " + se-
lectedTable);
            } else {
                System.out.println("No rows matched the filter.");
            }
        } catch (SQLException e) {
            System.err.println("Error executing update: " + e.getMessage());
            throw e;
        }
    }
```

```
    public void print() throws SQLException {
        Scanner scanner = new Scanner(System.in);
        List<String> tables = getTables("print");

        System.out.println("Choose a table to print data:");
        int tableIndex = scanner.nextInt();
        scanner.nextLine(); // Consume newline

        String selectedTable = tables.get(tableIndex);
        String query = "SELECT * FROM " + selectedTable;

        try (PreparedStatement preparedStatement = connection.prepareState-
ment(query);
             ResultSet resultSet = preparedStatement.executeQuery()) {

            ResultSetMetaData metaData = resultSet.getMetaData();
            int columnCount = metaData.getColumnCount();

            // Print header
            System.out.println("Printing data from: " + selectedTable);
            for (int i = 1; i <= columnCount; i++) {
                System.out.printf("%-20s", metaData.getColumnName(i)); // Fixed-
width formatting
            }
            System.out.println("\n" + "-".repeat(columnCount * 20));

            // Print rows
            while (resultSet.next()) {
                for (int i = 1; i <= columnCount; i++) {
                    System.out.printf("%-20s", resultSet.getString(i)); //
Fixed-width formatting
                }
                System.out.println();
            }
        } catch (SQLException e) {
            System.err.println("Error retrieving data from " + selectedTable +
": " + e.getMessage());
            throw e;
        }
    }
    /**
     * Execute an update/insert/delete query.
     */
    private void executeUpdate(String query, String successMessage) throws
SQLException {
        try (PreparedStatement preparedStatement = connection.prepareState-
ment(query)) {
            preparedStatement.executeUpdate();
            System.out.println(successMessage);
        } catch (SQLException e) {
            System.err.println("Error executing query: " + e.getMessage());
            throw e;
        }
    }

    /**
     * Generate SQL query for generation using PostgreSQL functions.
     */
    private String generateSQLForGeneration(String table, int count) {

        switch (table){
            case "users": return "INSERT INTO users (name) " +
                    "SELECT 'User' || trunc(random() * 100)::int FROM gener-
ate_series(1, " + count + ")";
```

```
            case "booking":
                return "INSERT INTO booking (users_id, trip_id, status, book-
ing_period, booking_desc)\n" +
                        "SELECT\n" +
                        "    u.users_id,\n" +
                        "    tr.trip_id,\n" +
                        "    (random() > 0.5) AS status,\n" +
                        "    daterange(\n" +
                        "        LEAST(date '2024-01-01' + d.day1, date '2024-
01-01' + d.day2),\n" +
                        "        GREATEST(date '2024-01-01' + d.day1, date
'2024-01-01' + d.day2)\n" +
                        "    ) AS booking_period,\n" +
                        "    'BookingDesc_' || trunc(random() * 100)::int AS
booking_desc\n" +
                        "FROM generate_series(1, " + count + ") g\n" +
                        "CROSS JOIN LATERAL (\n" +
                        "    -- Approximately 5 years * 365 ~ 1825 days\n" +
                        "    SELECT trunc(random() * 1825)::int AS day1,\n" +
                        "           trunc(random() * 1825)::int AS day2\n" +
                        ") d\n" +
                        "CROSS JOIN LATERAL (\n" +
                        "    SELECT users_id\n" +
                        "    FROM users\n" +
                        "    OFFSET floor(random() * (SELECT count(*) FROM us-
ers))\n" +
                        "    LIMIT 1\n" +
                        ") u\n" +
                        "CROSS JOIN LATERAL (\n" +
                        "    SELECT trip_id\n" +
                        "    FROM trip\n" +
                        "    OFFSET floor(random() * (SELECT count(*) FROM
trip))\n" +
                        "    LIMIT 1\n" +
                        ") tr;";
        }
        // Add cases for other tables as needed
        return "";
    }

    /**
     * Validate if a column is a foreign key.
     */
    private boolean isForeignKey(String column, String table) {
        // Identify foreign keys based on naming conventions
        return column.endsWith("_id") && !column.equals(table + "_id");
    }


    /**
     * Validate a foreign key exists in the referenced table.
     */
    private boolean validateForeignKey(String column, int value) throws SQLEx-
ception {
        // Correctly infer the referenced table name
        String referencedTable;
        if (column.equals("users_id")) {
            referencedTable = "users";
        } else if (column.equals("trip_id")) {
            referencedTable = "trip";
        } else {
            // Generic fallback for other foreign keys
            referencedTable = column.replace("_id", "");
        }

        String query = "SELECT 1 FROM " + referencedTable + " WHERE " + column +
```

```
" = ?";
        try (PreparedStatement statement = connection.prepareStatement(query)) {
            statement.setInt(1, value);
            try (ResultSet resultSet = statement.executeQuery()) {
                return resultSet.next();
            }
        }
    }


    /**
     * Validate date range format.
     */
    private boolean validateDateRange(String value) {
        return Pattern.matches("\\[[0-9]{4}-[0-9]{2}-[0-9]{2},[0-9]{4}-[0-9]{2}-
[0-9]{2}\\]", value);
    }

    /**
     * Get table names from the database schema.
     */
    public List<String> getTables(String operation) throws SQLException {
        List<String> tables = new ArrayList<>();
        String query = "SELECT table_name FROM information_schema.tables WHERE
table_schema = 'public' AND table_type = 'BASE TABLE'";

        try (PreparedStatement statement = connection.prepareStatement(query);
             ResultSet resultSet = statement.executeQuery()) {

            System.out.println("Choose a table to " + operation + " data:");
            int index = 0;
            while (resultSet.next()) {
                String tableName = resultSet.getString("table_name");
                tables.add(tableName);
                System.out.println(index + ": " + tableName);
                index++;
            }

            if (tables.isEmpty()) {
                System.out.println("No tables found in the database.");
            }
        } catch (SQLException e) {
            System.err.println("Error fetching tables: " + e.getMessage());
            throw e;
        }
        return tables;
    }


    /**
     * Get column names for a given table.
     */
    public List<String> getColumns(String table) throws SQLException {
        List<String> columns = new ArrayList<>();
        String query = "SELECT column_name FROM information_schema.columns WHERE
table_name = ?";
        try (PreparedStatement statement = connection.prepareStatement(query)) {
            statement.setString(1, table);
            try (ResultSet resultSet = statement.executeQuery()) {
                while (resultSet.next()) {
                    columns.add(resultSet.getString("column_name"));
                }
            }
        }
        return columns;
    }
```

```
    /**
     * Get column types for a given table.
     */
    public List<String> getColumnTypes(String table) throws SQLException {
        List<String> columnTypes = new ArrayList<>();
        String query = "SELECT data_type FROM information_schema.columns WHERE
table_name = ?";
        try (PreparedStatement statement = connection.prepareStatement(query)) {
            statement.setString(1, table);
            try (ResultSet resultSet = statement.executeQuery()) {
                while (resultSet.next()) {
                    columnTypes.add(resultSet.getString("data_type"));
                }
            }
        }
        return columnTypes;
    }
}
```

Опис:

1. **insert**:
   - Дозволяє вставляти дані у вибрану таблицю.
   - Перевіряє відповідність введених значень типу даних колонок.
2. **generate**:
   - Автоматично генерує випадкові дані для заданої таблиці(booking or users)
   - Використовує SQL-запити для генерації значень із дотриманням відповідності типів даних (наприклад, daterange).
   - Підраховує час виконання запиту.
3. **search**:
   - Реалізує пошук із фільтрацією (WHERE) і групуванням (GROUP BY) даних з кількох таблиць.
   - Дозволяє задавати умови фільтрації і групування з клавіатури.
   - Виводить результати запиту разом із часом його виконання.
4. **delete**:
   - Видаляє рядки з таблиці відповідно до умов фільтрації.
   - Перевіряє типи даних для коректного виконання запиту.
5. **update**:
   - Оновлює дані в таблиці за вказаними умовами.
   - Підтримує введення нових значень для оновлення рядків.
6. **print**:
   - Виводить усі дані з вибраної таблиці у форматованій таблиці.
   - Використовує фіксовану ширину колонок для читабельності.
7. **getTables**:
   - Отримує список усіх доступних таблиць у базі даних.
8. **getColumns**:
   - Отримує список усіх колонок вибраної таблиці.
9. **getColumnTypes**:
   - Отримує типи даних для кожної колонки вибраної таблиці.
10. **Валідація:**

- **isForeignKey**: Визначає, чи є колонка зовнішнім ключем.
- **validateForeignKey**: Перевіряє існування значення зовнішнього ключа у відповідній таблиці.
- **validateDateRange**: Валідує формат daterange.

11. **executeUpdate**:
    - Виконує запити типу INSERT, UPDATE, DELETE.