Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет прикладної математики

Кафедра системного програмування і спеціалізованих комп’ютерних систем

ЛАБОРАТОРНА РОБОТА № 2

з дисципліни

“Бази даних та засоби управління”

ТЕМА: “Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL”

Група: КВ-03
Виконав: Семенков М.С.
GitHub

Telegram

Оцінка:

Київ – 2022

Завдання на лабораторну роботу і вимоги до виконання

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

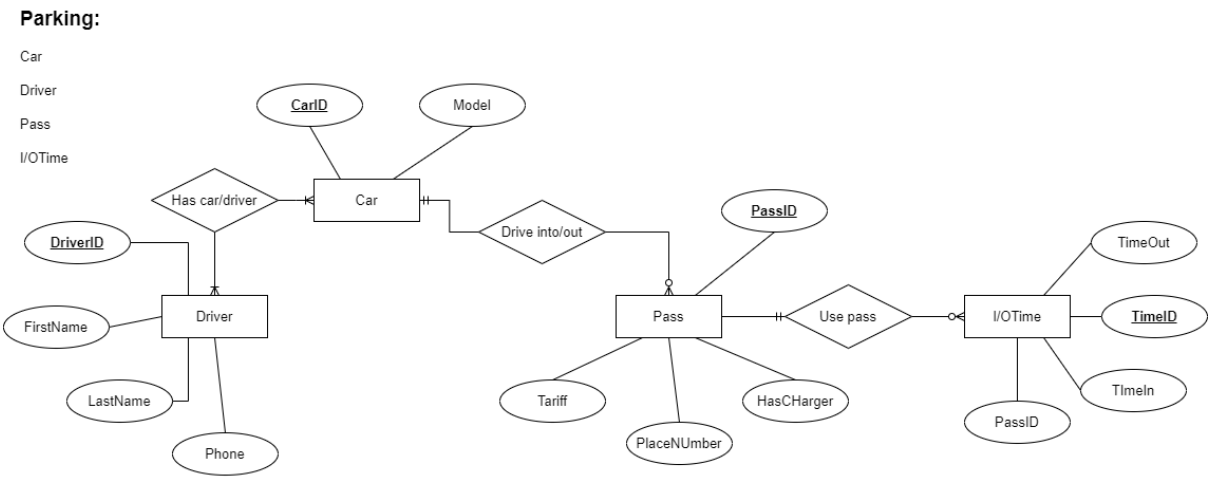
Загальне завдання роботи полягає у наступному:

1. Реалізувати функції перегляду, 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).

Опис та структура бази даних

Галузь розбита на 4 сутності: водій, машина, пропуск і час.

- Driver – сутність з описом водія, містить його ім’я, фамілію, унікальний ідентифікатор (номер прав) і контактний телефон. Ця сутність дозволяє ідентифікувати власників авто.
- Car – сутність з описом авто, містить назву моделі і унікальний ідентифікатор авто.
- Pass – сутність пропуска, містить його унікальний номер, ідентифікатор авто, до якого прив’язаний пропуск, номер місця на якому стане машина, інформацію про саме місце (наявність зарядки для електрокарів і тариф для цього місця). Ця сутність зв’язує авто з місцем на якому стоїть це авто.
- I/OTime – досить абстрактна сутність, призначена для нормалізації бази даних, це дозволяє не створювати нові записи при в’їзді/виїзді авто до/з паркінгу до таблиці сутності Pass. Ця сутність зберігає інформацію про час і ідентифікатор пропуску, до якого відносяться записи про час.



Діаграма моделі «сутність-зв'язок» предметної галузі "Паркінг" у нотації Чена

Схема меню користувача з описом функціональності кожного пункту

```
Choose action:
1 Read
2 Create
3 Update
4 Delete
5 Enable autocommit (default)
6 Disable autocommit
7 Commit
8 Rollback
9 Generate random values
10 Filter
11 close program
```

Меню користувача створеної програми

- Read – зчитування даних з однієї таблиці. Після обрання цього пункту, користувач може обрати потрібну таблицю і поля, що він хоче побачити. В результаті на екран будуть виведені обрані поля з обраної таблиці.
- Create – створення нового рядка в таблиці. Після обрання цього пункту, користувач може обрати таблицю, в яку він хоче ввести дані, після чого він має ввести самі дані. Якщо всі поля заповнені коректно, то новий рядок буде занесений до таблиці.

- Update – зміна вже існуючих даних. Після обрання цього пункту, користувач може обрати таблицю, в яку він хоче внести зміни, після цього він обирає поле, що хоче змінити і вводить нове значення, затім користувач вводить значення первинного ключа рядка, який він хоче змінити. Якщо введені дані валідні, то зміни будуть внесені у базу даних.
- Delete – видалення запису з таблиці. Після обрання цього пункту, користувач може обрати таблицю, в якій він хоче видалити запис, затім він вводить значення первинного ключа, що відповідає рядку, який користувач бажає видалити. Якщо ключ правильний, то відповідний рядок буде видалений.
- Enable autocommit – вмикає можливість автоматичного підтвердження змін. При запуску програми автокомміт за замовчуванням ввімкнено.
- Disable autocommit – вимикає можливість автоматичного підтвердження змін. В цьому випадку потрібно вручну це робити.
- Commit – підтвердження змін.
- Rollback – відкат до попереднього комміту.
- Generate random values – створення набору випадкових даних. Після обрання цього пункту, користувач може ввести кількість нових записів. Після цього автоматично у всіх таблицях буде створено відповідну кількість записів з урахуванням всіх зв’язків
- Filter – читання даних з декількох таблиць і їх фільтрація. Після обрання цього пункту, користувач може обрати один з трьох можливих варіантів комбінацій таблиць, затім він може ввести значення фільтрів відповідно до обраного пункту. В результаті будуть виведені записи, що відповідають вказаним фільтрам.
- Close program – завершення програми.

Використанні технічні засоби

Программа реалізована на мові програмування Java з використанням відповідного JDBC, створеного під PostgreSQL згідно з стандартами. JDBC дозволяє за допомогою відповідних команд формувати і виконувати запити до БД, а також фіксувати помилки, що приходять з неї.

Реалізація функції перегляду, внесення, редагування та вилучення даних у таблицях бази даних

car_id	model
1	Audi
2	Volkswagen
4	BMW
5	Mercedes Benz
3	Tesla

car_id	place_number	has_charger	tariff	pass_id
3	1	t	10	1
1	9	f	8	2
2	5	f	8	3
5	7	f	9	4

Вміст таблиць car і pass, що пов'язані завнішнім ключем

```
4
Choose table
1 car
2 cardriver
3 driver
4 i_otime
5 pass
1
Enter primary key value of row you want delete
3
ОШИБКА: UPDATE или DELETE в таблице "car" нарушает ограничение внешнего ключа "pass_fkey" таблицы "pass"
  Подробности: На ключ (car_id)=(3) всё ещё есть ссылки в таблице "pass".
```

Результат спроби видалення запису

Як видно, таблиці налаштовані так, що видалення полів, на які посилаються інші таблиці неможливо, відповідно спливає помилка. Адже з точки зору логіки галузі пропуск, що прив'язаний до певної машини не має сенсу, якщо інформації про цю машину нема.

Ідентична ситуація буде при спробі створення запису у таблиці, яка має зовнішній зв'язок до іншої, при не існуванні такого ключа. Знову ж таки, це випливає з логіки галузі, неможливо видати пропуск машині, про яку нічого не відомо.

```
2
Choose table
1 car
2 cardriver
3 driver
4 i_otime
5 pass
5
Set new values
Enter car_id
12
Enter place_number
10
Enter has_charger
false
Enter tariff
5
ОШИБКА: INSERT или UPDATE в таблице "pass" нарушает ограничение внешнего ключа "pass_fkey"
  Подробности: Ключ (car_id)=(12) отсутствует в таблице "car".
```

Результат спроби створення запису

Автоматичне пакетне генерування «рандомізованих» даних у базі

car_id	model
1	G
2	L
3	J
4	H
5	C
6	I
7	T
8	D
9	J
10	S
11	D
12	L
13	Y
14	I
15	E
16	Q
17	H
18	T
19	L
20	O

19979	R
19980	L
19981	F
19982	D
19983	W
19984	I
19985	I
19986	A
19987	I
19988	B
19989	L
19990	S
19991	V
19992	U
19993	J
19994	G
19995	A
19996	K
19997	X
19998	F
19999	F
20000	V

Результат генерації в таблиці car

driver_id	first_name	last_name	phone
1	JM	BR	69772
2	QY	FX	26500
3	QS	YU	36675
4	UT	DU	82215
5	HX	JU	88829
6	OO	DF	54292
7	SJ	XS	32305
8	KW	WO	75899
9	DS	OV	52055
10	JW	UT	19102
11	RK	OD	17848
12	SV	VL	91927
13	CY	GC	87627
14	HG	BH	71027
15	XB	CX	20802
16	RR	OX	41850
17	QN	UU	90723
18	NB	BN	28591
19	PC	QM	50878
20	MP	XY	29991

19980	BB	FJ	72935
19981	QA	HE	14630
19982	EW	WT	27168
19983	UT	QC	91606
19984	HX	PI	30093
19985	EM	OW	5132
19986	AG	IJ	23391
19987	PE	WT	70498
19988	BF	PJ	61572
19989	KW	LD	63963
19990	KT	FF	8749
19991	GW	XA	26384
19992	NW	HB	88312
19993	BT	BS	12226
19994	WU	RC	7109
19995	CC	EG	21640
19996	SR	KB	54209
19997	IU	QK	10082
19998	TS	RD	21338
19999	VO	CO	37426
20000	PI	OL	74749

Результат генерації в таблиці driver

car_id	driver_id
5622	12353
9891	2669
4628	19627
17232	15607
12367	10441
8702	2920
8925	17777
14651	8683
6414	10984
13175	10570
12989	17300
13719	9661
15267	16946
1874	14718
6718	15544
1569	13032
2145	2237
4846	8756
5346	7396
14944	422
19824	16089
4578	17269
19295	4804
3016	10658
6417	12399
13105	16383
4346	8634

Результат генерації в таблиці cardriver

car_id	place_number	has_charger	tariff	pass_id
10842	1	f	11	1
11063	2	f	15	2
18381	3	f	13	3
749	4	t	27	4
7439	5	t	23	5
11861	6	t	24	6
5754	7	t	25	7
19423	8	f	14	8
6277	9	f	19	9
15896	10	t	17	10
7297	11	f	28	11
7448	12	t	14	12
16147	13	f	27	13
2377	14	t	19	14
3291	15	t	18	15
6982	16	t	12	16
13382	17	t	22	17
7151	18	t	26	18
6651	19	f	23	19
8806	20	f	14	20

Результат генерації в таблиці pass

pass_id	time_in	time_out	time_id
7081	2022-02-15 15:00:00	2022-03-30 09:00:00	1
15839	2022-02-01 06:00:00	2022-03-19 12:00:00	2
15314	2022-02-01 22:00:00	2022-03-28 19:00:00	3
5502	2022-02-14 14:00:00	2022-03-25 18:00:00	4
2014	2022-01-11 17:00:00	2022-03-20 20:00:00	5
11378	2022-01-25 14:00:00	2022-02-22 20:00:00	6
14009	2022-02-14 09:00:00	2022-03-30 04:00:00	7
18777	2022-02-01 04:00:00	2022-03-10 18:00:00	8
9808	2022-01-19 18:00:00	2022-03-01 09:00:00	9
6559	2022-01-17 00:00:00	2022-03-17 22:00:00	10
6268	2022-02-15 18:00:00	2022-03-28 21:00:00	11
9620	2022-02-07 13:00:00	2022-02-22 05:00:00	12
5047	2022-02-18 17:00:00	2022-03-23 14:00:00	13
14165	2022-01-31 19:00:00	2022-03-30 08:00:00	14
10246	2022-02-12 04:00:00	2022-04-03 19:00:00	15
2247	2022-01-29 12:00:00	2022-03-25 07:00:00	16
15410	2022-02-15 02:00:00	2022-04-01 03:00:00	17
13021	2022-02-08 01:00:00	2022-03-12 01:00:00	18
5516	2022-01-21 07:00:00	2022-03-03 08:00:00	19
6738	2022-01-18 23:00:00	2022-03-28 23:00:00	20

Результат генерації в таблиці iotime

На жаль, згенерувати більше ніж 20 000 записів не вдалося, йе займає занадто багато часу. Можливо, якщо оптимізувати запити, то час виконання можна було б зменшити.

Лістинг SQL запиту для генерації псевдовипадкових записів:

```
DELETE FROM i_otime;

DELETE FROM pass;

DELETE FROM cardriver;

DELETE FROM car;

DELETE FROM driver;

INSERT INTO car(car_id, model)

SELECT car_id, chr(trunc(65 + random()*25)::INT) model

FROM generate_series(1, ?) car_id;

INSERT INTO driver(driver_id, first_name, last_name, phone)

SELECT driver_id, chr(trunc(65 + random()*25)::INT) || chr(trunc(65 +
random()*25)::INT) first_name,

chr(trunc(65 + random()*25)::INT) || chr(trunc(65 + random()*25)::INT)
first_name,

trunc(random() * 100000)::INT phone

FROM generate_series(1, ?) driver_id;

INSERT INTO cardriver(car_id, driver_id)

SELECT car_id, driver_id FROM generate_series(1, ?) car_id,
generate_series(1, ?) driver_id

ORDER BY random() LIMIT ?;

INSERT INTO pass(car_id, place_number, has_charger, tariff, pass_id)

SELECT 1 + random() * (? - 1), num, random() > 0.5 has_charger,
random() * 20 + 10 tariff, num

FROM generate_series(1, ?) num;

INSERT INTO i_otime(pass_id, time_in, time_out, time_id)

SELECT 1 + random() * (? - 1), TIMESTAMP '2022-01-10 20:00:00' +
trunc(random() * 1000)::INT * INTERVAL '1 hours',

TIMESTAMP '2022-02-21 12:00:00' + trunc(random() * 1000) * INTERVAL '1
hours', id

FROM generate_series(1, ?) id;
```

На місці знаків питання вставляється кількість генеруємих записів.

Реалізація пошуку за декількома атрибутами з двох та більше сутностей одночасно

```
10
1 Car, Driver
2 Pass, Car, Driver
3 IOTime, Pass, Car
1
Enter template for string model filter)
(% - any kind and num of symbols
_ - any kind one symbol
A
Enter template for string last_name filter)
(% - any kind and num of symbols
_ - any kind one symbol
%B
Range of driver id begins with
255
Range of driver id ends with
311
driver_id          | first_name         | last_name          | phone              | model              | car_id             |
268                | QU                 | IT                 | 73102              | A                  | 1470               |
294                | WJ                 | MN                 | 37164              | A                  | 3144               |
292                | JJ                 | FR                 | 29046              | A                  | 4300               |
270                | NH                 | AN                 | 67775              | A                  | 19951              |
272                | DM                 | JB                 | 47545              | P                  | 12574              |
266                | RH                 | PB                 | 56125              | A                  | 19166              |

Query executed in 20 ms
```

Лістинг SQL запиту (замість ? – параметри фільтрації, замість подвійних лапок– шаблон для LIKE)

```
SELECT driver_id, first_name, last_name, phone, model, car_id

FROM driver INNER JOIN cardriver USING(driver_id) INNER JOIN car USING(car_id)

WHERE (model LIKE '" "' OR last_name LIKE '" "') AND driver_id BETWEEN ? AND ? ;
```

```
10
1 Car, Driver
2 Pass, Car, Driver
3 IOTime, Pass, Car
2
Range of tariff begins with
5
Range of tariff ends with
12
Did place has charger (boolean true or false)
false
Range of place number begins with
118
Range of place number ends with
200
```

first_name	last_name	phone	model	place_number	tariff
YD	VJ	70467	C	1	11
NP	CI	1428	Y	39	10
HV	XK	60680	I	79	12
AV	JW	3396	T	83	11
JA	AQ	18529	M	106	10
FD	QI	78396	N	109	10
UG	CG	25369	Y	119	12
IA	AM	36579	H	125	11
DV	JB	9475	H	125	11
HA	JC	59452	H	125	11
SL	WV	56597	L	138	12
RM	VK	93859	P	147	11
XW	EI	39653	P	147	11

...

UG	QV	63861	B	19634	12	
UR	DE	62032	B	19650	12	
GD	FA	4789	K	19685	12	
GX	XT	95162	E	19746	12	
BV	FN	13801	W	19787	12	
GG	KY	73477	H	19798	12	
RI	DX	65674	F	19815	12	
HB	GQ	40477	U	19832	12	
WO	KI	26232	S	19837	12	
II	AR	32660	J	19884	10	
BB	NS	9207	U	19895	10	
DJ	AP	95652	U	19895	10	
VS	QC	12681	B	19899	12	
HA	QA	18709	W	19911	11	
QE	JM	63727	W	19911	11	
IX	YR	99789	W	19914	11	
NL	IG	81065	D	19925	11	
YC	AX	38739	W	19929	10	
VQ	CJ	19429	J	19961	11	
MN	QS	29271	B	19997	10	
WT	DY	21301	B	19997	10	
SJ	SK	29894	V	19999	11	

```
Query executed in 263 ms
```

Лістинг SQL запиту (замість ? – параметри фільтрації)

```
SELECT first_name, last_name, phone, model, place_number, tariff

FROM pass INNER JOIN car USING(car_id)

INNER JOIN cardriver USING(car_id)

INNER JOIN driver USING(driver_id)

WHERE tariff BETWEEN ? AND ? AND (has_charger = ? OR place_number
BETWEEN ? AND ?);ver_id BETWEEN ? AND ? ;
```

```
10
1 Car, Driver
2 Pass, Car, Driver
3 IOTime, Pass, Car
3
Range of time in begins with
2022-02-01 10:00:00
Range of time in ends with
2022-02-04 18:00:00
Did place has charger (boolean true or false)
true
Enter template for string model filter)
(% - any kind and num of symbols
_ - any kind one symbol
V
```

car_id	model	place_number	tariff	time_in	time_out
14069	W	12368	16	2022-02-03 06:00:00	2022-03-13 18:00:00
2094	V	14469	16	2022-02-03 11:00:00	2022-03-14 06:00:00
236	O	11944	23	2022-02-01 12:00:00	2022-03-17 14:00:00
3524	G	8638	20	2022-02-03 15:00:00	2022-03-15 14:00:00
16807	J	15202	27	2022-02-03 22:00:00	2022-03-20 12:00:00
9100	C	12149	16	2022-02-04 09:00:00	2022-03-30 10:00:00
4725	V	5650	30	2022-01-16 08:00:00	2022-03-03 07:00:00
17406	V	11863	24	2022-02-02 22:00:00	2022-03-08 09:00:00
6065	W	4981	21	2022-02-04 01:00:00	2022-04-02 15:00:00
18954	V	8538	26	2022-02-02 21:00:00	2022-03-02 04:00:00
19844	A	19355	30	2022-02-03 16:00:00	2022-04-01 04:00:00
18989	D	12846	27	2022-02-01 10:00:00	2022-03-13 09:00:00

...

3885	V	14227	20	2022-02-03 10:00:00	2022-03-08 01:00:00
689	R	7233	21	2022-02-02 00:00:00	2022-03-15 21:00:00
8777	V	8285	12	2022-02-03 06:00:00	2022-03-27 07:00:00
19125	V	4027	16	2022-01-30 10:00:00	2022-02-24 00:00:00
8673	H	19244	24	2022-02-02 14:00:00	2022-03-19 22:00:00
1448	V	5995	22	2022-01-12 19:00:00	2022-03-09 03:00:00
4610	V	17652	26	2022-02-01 21:00:00	2022-02-25 13:00:00
11054	V	10446	23	2022-01-12 04:00:00	2022-02-26 23:00:00
10773	V	9491	14	2022-01-23 15:00:00	2022-03-28 01:00:00
9263	V	18976	28	2022-02-19 03:00:00	2022-02-23 04:00:00
377	V	3316	10	2022-01-25 07:00:00	2022-02-22 01:00:00
12089	V	4888	28	2022-02-06 21:00:00	2022-03-17 02:00:00
15086	F	7725	15	2022-02-04 02:00:00	2022-03-31 08:00:00

```
Query executed in 263 ms
```

Лістинг SQL запиту (замість ? – параметри фільтрації, замість подвійних лапок– шаблон для LIKE)

```
SELECT car_id, model, place_number, tariff, time_in, time_out

FROM i_otime INNER JOIN pass USING(pass_id)

INNER JOIN car USING(car_id)

WHERE time_in::TIMESTAMP BETWEEN ? AND ? AND has_charger = ? OR model
LIKE '" "';
```

Реалізація шаблону MVC

Метод Model()

```
Model() {

        connection = getConnection();

    }
```

Конструктор, при створенні нового об’єкту пробує під’єднатися до бази даних.

Метод getTableList()

```
List<String> getTablesList() {

        List<String> tablesList = new LinkedList<>();

        try (Statement stmt = connection.createStatement();

             ResultSet rs = stmt.executeQuery(SQLCommands.sqlSelectTables)) {

            while (rs.next()) {

                tablesList.add(rs.getString(1));

            }

        } catch (SQLException ex) {

            System.out.println(ex.getMessage());

        }

        return tablesList;

    }
```

Метод повертає список назв таблиць, що знаходяться в базі даних.

Метод getColumnsList(...)

```
List<String> getColumnsList(String tableName, boolean fullList) {

        List<String> tablesList = new LinkedList<>();

        try (Statement stmt = connection.createStatement();

             ResultSet rs = stmt.executeQuery(SQLCommands.sqlSelectColumns[0]
+ tableName +

                     SQLCommands.sqlSelectColumns[(fullList) ? 2 : 1])) {
```

```
            while (rs.next()) {
                tablesList.add(rs.getString(1));
            }
        } catch (SQLException ex) {
            System.out.println(ex.getMessage());
        }
        return tablesList;
    }
```

Метод повертає список колонок, з яких складається таблиця. Якщо параметр fullList визначений, як false, то він повертає лише колонки без автогенерації.

Метод read(...)

```
List<List<String>> read(String params, String tableName, int amountOfColumns)
{
        List<List<String>> tablesList = new LinkedList<>();
        try (Statement stmt = connection.createStatement();
             ResultSet rs = getSelectResultSet(stmt, params, tableName)) {
            tablesList.add(new LinkedList<>());
            int lineNum = 0;
            while (rs.next()) {
                for (int i = 1; i <= amountOfColumns; i++) {
                    tablesList.get(lineNum).add(rs.getString(i));
                }
                tablesList.add(new LinkedList<>());
                lineNum++;
            }
        } catch (SQLException ex) {
            System.out.println(ex.getMessage());
        }
        return tablesList;
```

```
    }
```

Метод повертає список зі значенням вказаних колонок, вказаної таблиці.

Метод getSelectResultSet(...)

```
ResultSet getSelectResultSet(Statement stmt, String params, String tableName)
throws SQLException {

        return stmt.executeQuery(SQLCommands.sqlSelect[0] + params +
SQLCommands.sqlSelect[1] + tableName + SQLCommands.sqlSelect[2]);

    }
```

Метод підготовлює запит для читання полів таблиці.

Метод readWithFilter(...)

```
public List<List<String>> readWithFilter(int menuNum, List<String> params) {

        List<List<String>> result = new LinkedList<>();

        try (PreparedStatement ps = readWithFilterPrepareStatement(menuNum,
params);

             ResultSet rs = ps.executeQuery()) {

//             System.out.println(ps);

            result.add(new LinkedList<>());

            int lineNum = 0;

            int amountOfColumns = 6;

            while (rs.next()) {

                for (int i = 1; i <= amountOfColumns; i++) {

                    result.get(lineNum).add(rs.getString(i));

                }

                result.add(new LinkedList<>());

                lineNum++;

            }

        } catch (SQLException ex) {

            System.out.println(ex.getMessage());

        }
```

```
        return result;

    }
```

Метод повертає список із значеннями, що є результатом фільтрації декартового добутку декількох таблиць.

Метод readWithFilterPrepareStatement(...)

```
private PreparedStatement readWithFilterPrepareStatement(int menuNum,
List<String> params) throws SQLException {

        PreparedStatement ps = null;

        switch (menuNum) {

            case 1 -> {

                ps =
connection.prepareStatement(SQLCommands.sqlSelectFilterCarDriver[0] +
params.get(0) +

                        SQLCommands.sqlSelectFilterCarDriver[1] +
params.get(1) +

                        SQLCommands.sqlSelectFilterCarDriver[2]);

                ps.setInt(1, Integer.parseInt(params.get(2)));

                ps.setInt(2, Integer.parseInt(params.get(3)));

            }

            case 2 -> {

                ps =
connection.prepareStatement(SQLCommands.sqlSelectFilterPassCarDriver);

                ps.setInt(1, Integer.parseInt(params.get(0)));

                ps.setInt(2, Integer.parseInt(params.get(1)));

                ps.setBoolean(3, Boolean.parseBoolean(params.get(2)));

                ps.setInt(4, Integer.parseInt(params.get(3)));

                ps.setInt(5, Integer.parseInt(params.get(4)));

            }

            case 3 -> {

                ps =
connection.prepareStatement(SQLCommands.sqlSelectFilterIOTimePassCar[0] +
params.get(3) +
```

```
                        SQLCommands.sqlSelectFilterIOTimePassCar[1]);
                ps.setTimestamp(1, Timestamp.valueOf(params.get(0)));
                ps.setTimestamp(2, Timestamp.valueOf(params.get(1)));
                ps.setBoolean(3, Boolean.parseBoolean(params.get(2)));
            }
            default -> {
                throw new SQLException("Wrong menu number");
            }
        }
        return ps;
    }
```

Метод підготовлює запит для фільтрації декартового добутку декількох таблиць.

Метод create(...)

```
int create(String tableName, List<String> values) {
        int result = -1;
        try (PreparedStatement ps = createPrepareStatement(tableName,
values)) {
            result = ps.executeUpdate();
        } catch (SQLException ex) {
            System.out.println(ex.getMessage());
        }
        return result;
    }
```

Метод створює запис у заданій таблиці з заданими значеннями.

Метод createPrepareStatement(...)

```
private PreparedStatement createPrepareStatement(String tableName,
List<String> values) throws SQLException {
        PreparedStatement ps = null;
```

```
        switch (tableName) {
            case "car" -> {
                ps = connection.prepareStatement(SQLCommands.sqlInsertInCar);
                ps.setInt(1, Integer.parseInt(values.get(0)));
                ps.setString(2, values.get(1));
            }
            case "driver" -> {
                ps =
connection.prepareStatement(SQLCommands.sqlInsertInDriver);
                ps.setInt(1, Integer.parseInt(values.get(0)));
                ps.setString(2, values.get(1));
                ps.setString(3, values.get(2));
                ps.setString(4, values.get(3));
            }
            case "cardriver" -> {
                ps =
connection.prepareStatement(SQLCommands.sqlInsertInCarDriver);
                ps.setInt(1, Integer.parseInt(values.get(0)));
                ps.setInt(2, Integer.parseInt(values.get(1)));
            }
            case "pass" -> {
                ps =
connection.prepareStatement(SQLCommands.sqlInsertInPass);
                ps.setInt(1, Integer.parseInt(values.get(0)));
                ps.setInt(2, Integer.parseInt(values.get(1)));
                ps.setBoolean(3, Boolean.parseBoolean(values.get(2)));
                ps.setInt(4, Integer.parseInt(values.get(3)));
            }
            case "i/otime" -> {
                ps =
connection.prepareStatement(SQLCommands.sqlInsertInIOTime);
```

```
                ps.setInt(1, Integer.parseInt(values.get(0)));
                ps.setTimestamp(2, Timestamp.valueOf(values.get(1)));
                ps.setTimestamp(3, Timestamp.valueOf(values.get(2)));
            }
            default -> {
                throw new SQLException("Wrong table name");
            }
        }
        return ps;
    }
```

Метод підготовлює запит для створення нового запису в таблиці.

Метод update(...)

```
int update(String tableName, String changeValue, String newValue, String[]
filterValues) {
        int result = -1;
        try (PreparedStatement ps = updatePrepareStatement(tableName,
changeValue, newValue, filterValues)) {
            result = ps.executeUpdate();
        } catch (SQLException ex) {
            System.out.println(ex.getMessage());
        }
        return result;
    }
```

Метод змінює вже існуючий запис у заданій таблиці, необхідний запис визначається значенням первинного ключа.

Метод updatePrepareStatement(...)

```
private PreparedStatement updatePrepareStatement(String tableName, String
changeValue, String newValue, String[] filterValues) throws SQLException {
        PreparedStatement ps = null;
        switch (tableName) {
```

```
            case "car" -> {
                switch (changeValue) {
                    case "car_id" -> {
                        ps =
connection.prepareStatement(SQLCommands.sqlUpdateCarID);
                        ps.setInt(1, Integer.parseInt(newValue));
                        ps.setInt(2, Integer.parseInt(filterValues[0]));
                    }
                    case "model" -> {
                        ps =
connection.prepareStatement(SQLCommands.sqlUpdateCarModel);
                        ps.setString(1, newValue);
                        ps.setInt(2, Integer.parseInt(filterValues[0]));
                    }
                }
            }
            case "driver" -> {
                switch (changeValue) {
                    case "driver_id" -> {
                        ps =
connection.prepareStatement(SQLCommands.sqlUpdateDriverID);
                        ps.setInt(1, Integer.parseInt(newValue));
                        ps.setInt(2, Integer.parseInt(filterValues[0]));
                    }
                    case "first_name" -> {
                        ps =
connection.prepareStatement(SQLCommands.sqlUpdateDriverFirstName);
                        ps.setString(1, newValue);
                        ps.setInt(2, Integer.parseInt(filterValues[0]));
                    }
                    case "last_name" -> {
```

```
                        ps =
connection.prepareStatement(SQLCommands.sqlUpdateDriverLastName);

                        ps.setString(1, newValue);

                        ps.setInt(2, Integer.parseInt(filterValues[0]));

                    }

                    case "phone" -> {

                        ps =
connection.prepareStatement(SQLCommands.sqlUpdateDriverPhone);

                        ps.setString(1, newValue);

                        ps.setInt(2, Integer.parseInt(filterValues[0]));

                    }

                }

            }

            case "cardriver" -> {

                switch (changeValue) {

                    case "car_id" -> {

                        ps =
connection.prepareStatement(SQLCommands.sqlUpdateCarDriverCarID);

                        ps.setInt(1, Integer.parseInt(newValue));

                        ps.setInt(2, Integer.parseInt(filterValues[0]));

                        ps.setInt(3, Integer.parseInt(filterValues[1]));

                    }

                    case "driver_id" -> {

                        ps =
connection.prepareStatement(SQLCommands.sqlUpdateCarDriverDriverID);

                        ps.setInt(1, Integer.parseInt(newValue));

                        ps.setInt(2, Integer.parseInt(filterValues[0]));

                        ps.setInt(3, Integer.parseInt(filterValues[1]));

                    }

                }

            }
```

```
            case "pass" -> {

                switch (changeValue) {

                    case "pass_id" -> {

                        ps =
connection.prepareStatement(SQLCommands.sqlUpdatePassID);

                        ps.setInt(1, Integer.parseInt(newValue));

                        ps.setInt(2, Integer.parseInt(filterValues[0]));

                    }

                    case "has_charger" -> {

                        ps =
connection.prepareStatement(SQLCommands.sqlUpdatePassHasCharger);

                        ps.setBoolean(1, Boolean.parseBoolean(newValue));

                        ps.setInt(2, Integer.parseInt(filterValues[0]));

                    }

                    case "place_number" -> {

                        ps =
connection.prepareStatement(SQLCommands.sqlUpdatePassPlaceNumber);

                        ps.setInt(1, Integer.parseInt(newValue));

                        ps.setInt(2, Integer.parseInt(filterValues[0]));

                    }

                    case "tariff" -> {

                        ps =
connection.prepareStatement(SQLCommands.sqlUpdatePassTariff);

                        ps.setInt(1, Integer.parseInt(newValue));

                        ps.setInt(2, Integer.parseInt(filterValues[0]));

                    }

                    case "car_id" -> {

                        ps =
connection.prepareStatement(SQLCommands.sqlUpdatePassCarID);

                        ps.setInt(1, Integer.parseInt(newValue));

                        ps.setInt(2, Integer.parseInt(filterValues[0]));

                    }
```

```
                }
            }
            case "i/otime" -> {
                switch (changeValue) {
                    case "time_id" -> {
                        ps =
connection.prepareStatement(SQLCommands.sqlUpdateIOTimeID);
                        ps.setInt(1, Integer.parseInt(newValue));
                        ps.setInt(2, Integer.parseInt(filterValues[0]));
                    }
                    case "pass_id" -> {
                        ps =
connection.prepareStatement(SQLCommands.sqlUpdateIOTimePassID);
                        ps.setInt(1, Integer.parseInt(newValue));
                        ps.setInt(2, Integer.parseInt(filterValues[0]));
                    }
                    case "time_in" -> {
                        ps =
connection.prepareStatement(SQLCommands.sqlUpdateIOTimeIn);
                        ps.setInt(1, Integer.parseInt(newValue));
                        ps.setTimestamp(2,
Timestamp.valueOf(filterValues[0]));
                    }
                    case "time_out" -> {
                        ps =
connection.prepareStatement(SQLCommands.sqlUpdateIOTimeOut);
                        ps.setInt(1, Integer.parseInt(newValue));
                        ps.setTimestamp(2,
Timestamp.valueOf(filterValues[0]));
                    }
                }
            }
```

```
            default -> throw new SQLException("Wrong table name");
        }
        return ps;
    }
```

Метод формує запит для зміни запису.

Метод delete(...)

```
int delete(String tableName, String[] primaryKeyValues) {
        int result = -1;
        try (PreparedStatement ps = deletePrepareStatement(tableName)) {
            ps.setInt(1, Integer.parseInt(primaryKeyValues[0]));
            if (tableName.equals("cardriver")) {
                ps.setInt(2, Integer.parseInt(primaryKeyValues[1]));
            }
            result = ps.executeUpdate();
        } catch (SQLException ex) {
            System.out.println(ex.getMessage());
        }
        return result;
    }
```

Метод видаляє запис із вказаної таблиці, відповідно до значення первинного ключа.

Метод deletePrepareStatement(...)

```
private PreparedStatement deletePrepareStatement(String tableName) throws
SQLException {
        PreparedStatement ps = null;
        switch (tableName) {
            case "car" -> ps =
connection.prepareStatement(SQLCommands.sqlDeleteCar);
            case "driver" -> ps =
connection.prepareStatement(SQLCommands.sqlDeleteDriver);
```

```
            case "cardriver" -> ps =
connection.prepareStatement(SQLCommands.sqlDeleteCarDriver);

            case "pass" -> ps =
connection.prepareStatement(SQLCommands.sqlDeletePass);

            case "i/otime" -> ps =
connection.prepareStatement(SQLCommands.sqlDeleteIOTime);

            default -> throw new SQLException("Wrong table name");

        }

        return ps;

    }
```

Метод формує запит для видалення запису із таблиці.

Метод generateRows(...)

```
int generateRows(int rowsAmount) {

        int result = -1;

        try (PreparedStatement ps =
connection.prepareStatement(SQLCommands.sqlGenerateRows)) {

            ps.setInt(1, rowsAmount);

            ps.setInt(2, rowsAmount);

            ps.setInt(3, rowsAmount);

            ps.setInt(4, rowsAmount);

            ps.setInt(5, rowsAmount);

            ps.setInt(6, rowsAmount);

            ps.setInt(7, rowsAmount);

            ps.setInt(8, rowsAmount);

            ps.setInt(9, rowsAmount);

            result = ps.executeUpdate();

        } catch (SQLException ex) {

            System.out.println(ex.getMessage());

        }

        return result;

    }
```

Метод формує і виконує запит для генерації вказаної кількості псевдовипадкових значень у всіх таблицях.

Метод getConnection()

```
Connection getConnection() {

        try {

            String[] connectionInfo = getConnectionInfo();

            return DriverManager.getConnection(connectionInfo[0],
connectionInfo[1], connectionInfo[2]);

        } catch (SQLException ex) {

            System.out.println("Can not connect to data base");

        } catch (IOException ex) {

            System.out.println("Can not open file");

        }

        return null;

    }
```

Метод повертає підключення до бази даних. Пароль, логін і адреса підключення беруться з конфігураційного файла.

Метод getConnectionInfo()

```
private String[] getConnectionInfo() throws IOException {

        String[] connectionInfo = new String[3];

        try (FileReader fileReader = new
FileReader(Main.CONN_INFO_FILE_LOCATION);

             Scanner scanner = new Scanner(fileReader)) {

            while (scanner.hasNext()) {

                String line = scanner.next();

                if (line.equals("URL:")) {

                    connectionInfo[0] = scanner.next();

                } else if (line.equals("USER:")) {

                    connectionInfo[1] = scanner.next();

                } else if (line.equals("PASSWORD:")) {
```

```
                    connectionInfo[2] = scanner.next();
                }
            }
        }
        return connectionInfo;
    }
```

Метод дістає з конфігураційного файлу інформацію, необхідну для підключення до бази даних.

Метод desableAutocommit()

```
void disableAutocommit() {
        try {
            connection.setAutoCommit(false);
        } catch (SQLException ex) {
            System.out.println("Can not change autocommit status");
        }
    }
```

Метод вимикає автоматичне підтвердження змін.

Метод enableAutocommit()

```
    void enableAutocommit() {
        try {
            connection.setAutoCommit(true);
        } catch (SQLException ex) {
            System.out.println("Can not change autocommit status");
        }
    }
```

Метод вмикає автоматичне підтвердження змін.

Метод commit()

```
    void commit() {
```

```
        try {
            connection.commit();
        } catch (SQLException e) {
            System.out.println("Can not commit changes");
        }
    }
```

Метод підтверджує зміни.

Метод rollback()

```
    void rollback() {
        try {
            connection.rollback();
        } catch (SQLException e) {
            System.out.println("Can not rollback");
        }
    }
```

Метод відміняє всі зміни до попереднього комміту.

Метод closeConnection()

```
    void closeConnection() {
        try {
            connection.close();
        } catch (SQLException ex) {
            System.out.println("Can not close connection");
        }
    }
```

Метод закриває підключення до бази даних.