

Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут»

Розрахунково-графічна робота
з дисципліни «Бази даних та засоби управління»

**«Створення додатку бази даних, орієнтованого на
взаємодію з СУБД PostgreSQL»**

Виконав студент групи: КВ-11

ПІБ: Терентьєв І.Д.

Телеграм: @t3ry444y

Репозиторій: [посилання](#)

Перевірив: _____

Київ 2023

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).

Варіант(предметна область): Система управління донорами крові.

Звіт

Опис бази даних

При донорстві крові важливо враховувати тип крові, час його здачі, дані про донора та реципієнта. Один донор може здати кров декілька разів, й реципієнт може потребувати декілька пакетів крові. Один пакет крові може бути отриманий як в результаті однієї здачі крові, так і декількох.

В моделі предметної області(рис. 1), що задана, виділяються наступні сутності та зв'язки між ними:

- Сутність “Donor” з атрибутами: DonorID, FirstName, LastName, DateOfBirth, ContactNumber, BloodType
 - Сутність “BloodBank” з атрибутами: BloodBankID, Name, Location, ContactNumber, TotalDonations
 - Сутність “BloodDonation” з атрибутами: DonationID, DonationDate, DonationTime, DonorID, DonationStatus
 - Сутність “BloodBag” з атрибутами: BagID, BloodType, ExpiryDate, StorageTemperature
 - Сутність “Recipient” з атрибутами: RecipientID, FirstName, LastName, DateOfBirth, ContactNumber, BloodTypeNeeded
- ☐ Між сутностями “Donor” та “BloodDonation” зв’язок 1:N, тому що здач крові від одного донора може бути декілька.
 - ☐ Між сутностями “BloodBank” та “BloodDonation” зв’язок 1:N, тому що здач крові може бути багато, але лише в один банк крові.
 - ☐ Між сутностями “BloodBag” та “BloodDonation” зв’язок M:N, тому що декілька здач крові можуть бути використані для створення одного пакета крові, так само одна здача крові може бути використана для декількох пакетів крові.

- Між сутностями “BloodBag” та “Recipient” зв’язок 1:N, тому що один реципієнт може потребувати декілька пакетів крові, але один реципієнт потребує щонайменше одного пакета крові.

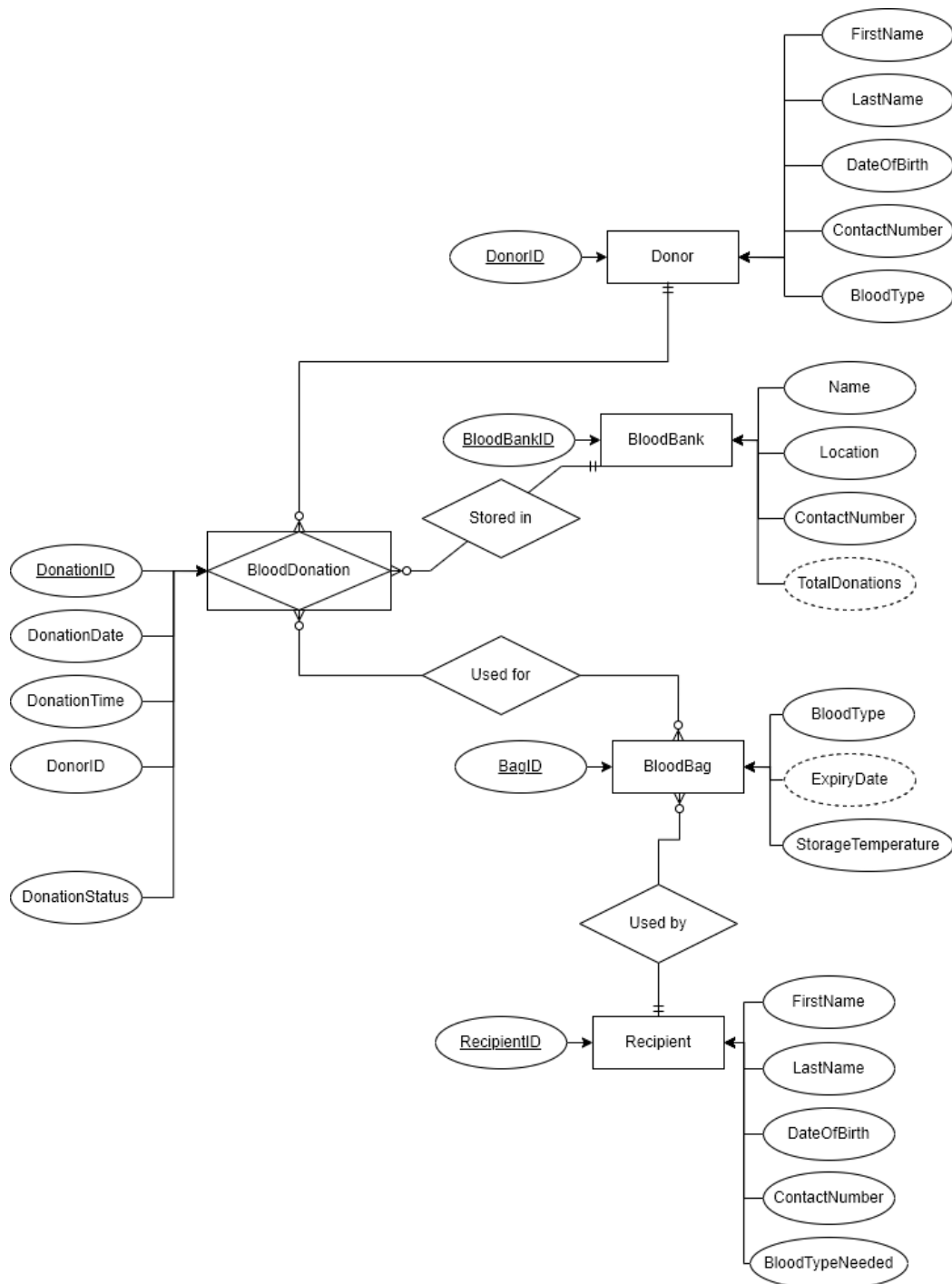


Рисунок 1. ER-модель предметної області “Система управління донорами крові”, створена на сервісі draw.io. Використана нотація “Crow’s foot”.

Схема бази даних PostgreSQL на основі ER-моделі обраної предметної області(рис. 2).

- Сутність “Donor” було перетворено в таблицю “Donor”
- Сутність “BloodBank” було перетворено в таблицю “BloodBank”
- Сутність “Recipient” було перетворено в таблицю “Recipient”
- Сутність “BloodDonation” було перетворено в таблицю “BloodDonation”
- Сутність “BloodBag” було перетворено в таблицю “BloodBag”
- Зв’язок “Used for” M:N між “BloodBag” та “BloodDonation” призвів до створення таблиці “BloodBag-BloodDonation” та двох полів “BloodDonationID” та “BloodBagID”
- Зв’язок “Stored in” 1:N призвів до появи додаткового поля “BloodDonationID” в “BloodBank”
- Зв’язок “Used by” 1:N призвів до появи додаткового поля “BloodBagID” в “Recipient”

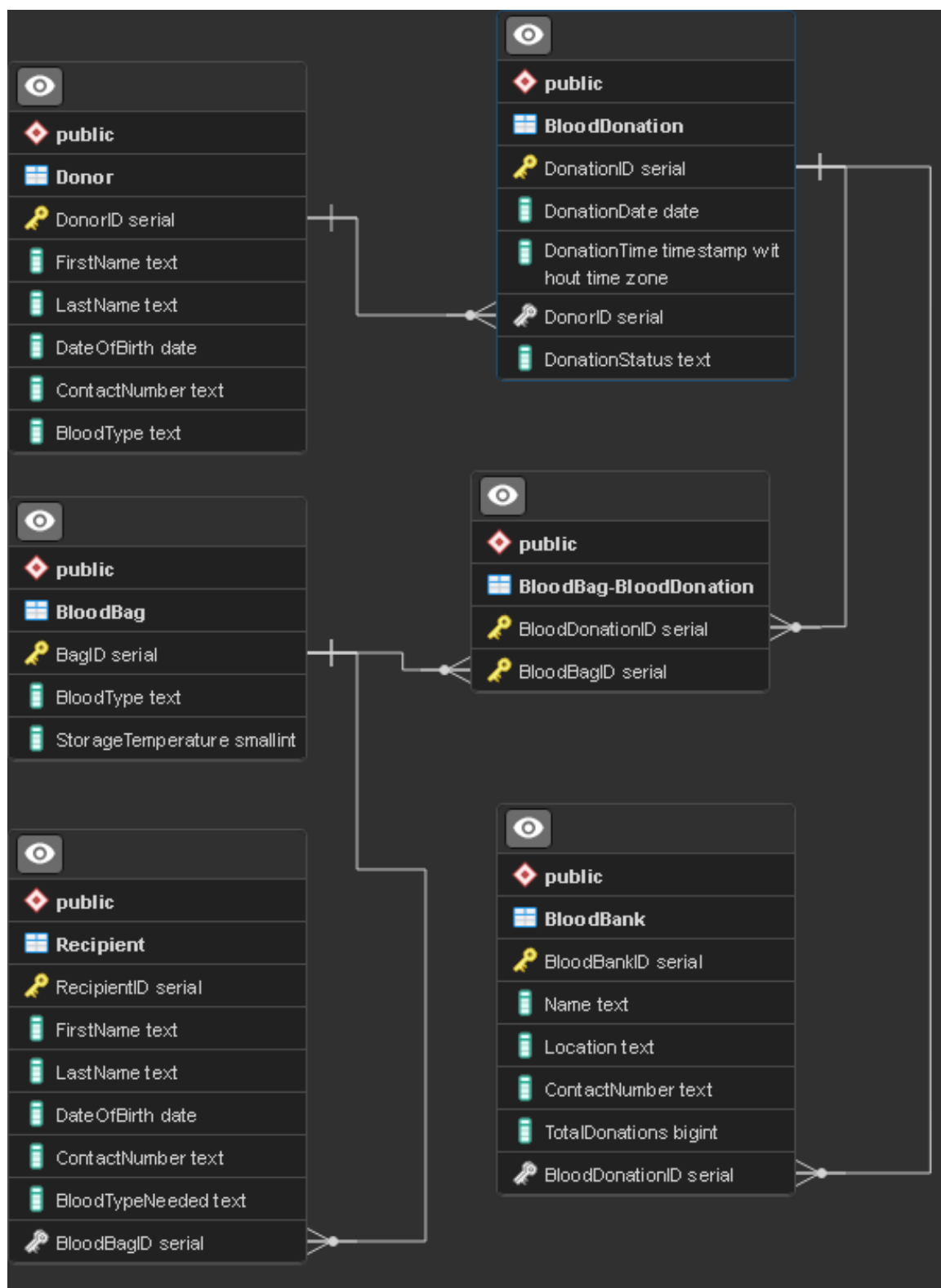


Рисунок 2. Схема бази даних PostgreSQL

Схема меню користувача, опис функціональності кожного пункту

```
Menu:
1) View table
2) View all tables
3) Add to table
4) Delete from table
5) Edit table
6) Randomize table
7) Search
8) Exit
Input number:
>?
```

- 1) View table - перегляд однієї таблиці з бази даних, користувач обирає таблицю
- 2) View all tables - перегляд усіх таблиць з бази даних
- 3) Add to table - додає рядок до бази даних, користувач вводить дані для кожного стовпчика
- 4) Delete from table - видаляє рядок по введеному користувачем id
- 5) Edit table - користувач вибирає рядок з таблиці та редагує дані одного з стовпчиків на вибір
- 6) Randomize table - додає рядок до бази даних, дані рядка рандомізовані, користувач вводить кількість рядків, що будуть додані до обраної таблиці
- 7) Search - користувач вибирає один з пошукових запитів, вводить константи пошуку, отримує таблицю результат та час виконання пошуку

Назва мови програмування та використані бібліотеки

Мова: Python

Використані бібліотеки, з файлу requirements.txt:

prettytable~=3.9.0

phonenumbers~=8.13.26

Faker~=20.1.0

psycopg2-binary~=2.9.9

Завдання №1

```
Input number: >? 1
Select table:
1) BloodBag
2) BloodBag-BloodDonation
3) BloodBank
4) BloodDonation
5) Donor
6) Recipient
Input number: >? 1
```

BagID	BloodType	StorageTemperature
21	A-	20
22	O+	20
23	A+	18
24	A-	19
25	A-	18
26	O-	18
27	B-	20
28	AB+	19
29	B+	18
30	AB+	18

Операція перегляду таблиці

Операція вилучення з результатом перехоплення та подальшим виведенням про неможливість виконання операції

1SELECT * FROM "BloodBag-BloodDonation" INNER JOIN "BloodBag" ON "BagID"="BloodBagID"

Data OutputMessagesNotifications

	BloodDonationID integer	BloodBagID integer	BagID integer	BloodType text	StorageTemperature smallint
1	5585	275	275	B+	20
2	2865	64	64	B-	18
3	5719	217	217	B-	18
4	2114	954	954	O+	18
5	2922	950	950	B+	20
6	7779	936	936	A-	18
7	2090	593	593	O+	19
8	7273	511	511	O-	18
9	3292	671	671	A-	20
10	3488	931	931	AB-	19

```
Select id to delete:
Enter id: >? 64
Got error from server:
ПОМИЛКА: update або delete в таблиці "BloodBag" порушує обмеження зовнішнього ключа "BloodBagID" таблиці "BloodBag-BloodDonation"
DETAIL:  На ключ (BagID)=(64) все ще є посилання в таблиці "BloodBag-BloodDonation".
```

Операція вставки запису з результатом перехоплення та подальшим виведенням про неможливість виконання операції.

1SELECT * FROM public."BloodBag"
2ORDER BY "BagID" DESC LIMIT 100
3

Data OutputMessagesNotifications

	BagID [PK] integer	BloodType text	StorageTemperature smallint
1	1030	B+	19
2	1029	O-	20
3	1028	O+	20
4	1027	A+	18









































```
Input params:
1) BloodDonationID
Input value: >? 5789
2) BloodBagID
Input value: >? 1031
Got error from server:
ПОМИЛКА: insert або update в таблиці "BloodBag-BloodDonation" порушує обмеження зовнішнього ключа "BloodBagID"
DETAIL:  Ключ (BloodBagID)=(1031) не присутній в таблиці "BloodBag".
```

Завдання №2

Генерація 100000 рандомізованих рядків в таблиці Donor

```
Select table:
1) BloodBag
2) BloodBag-BloodDonation
3) BloodBank
4) BloodDonation
5) Donor
6) Recipient
Input number: >? 6

+-----+-----+-----+-----+-----+-----+-----+
| RecipientID | FirstName | LastName | DateOfBirth | ContactNumber | BloodTypeNeeded | BloodBagID |
+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+
Input count of random elements: >? 100000
```

Query		Query History
1	select count("DonorID") from "Donor"	
Data Output		Messages
		Notifications
		
		
		
		
		
		
		
		
		
		
		
		
		
		

1SELECT * FROM public."Donor"

2ORDER BY "DonorID" DESC LIMIT 100

3

Data Output

Messages

Notifications

<

Генерація 100000 рандомізованих рядків в таблиці Recipient

```
Select table:
1) BloodBag
2) BloodBag-BloodDonation
3) BloodBank
4) BloodDonation
5) Donor
6) Recipient
Input number: >? 6
+-----+-----+-----+-----+-----+-----+-----+
| RecipientID | FirstName | LastName | DateOfBirth | ContactNumber | BloodTypeNeeded | BloodBagID |
+-----+-----+-----+-----+-----+-----+-----+
Input count of random elements: >? 100000
```

1 select count("RecipientID") from "Recipient"

Data Output Messages Notifications

count
bigint

1

100000

1 SELECT * FROM public."Recipient"
2 ORDER BY "RecipientID" DESC LIMIT 100
3

Data Output Messages Notifications

	RecipientID [PK] integer	FirstName text	LastName text	DateOfBirth date	ContactNumber text	BloodTypeNeeded text	BloodBagID integer
1	200042	Kimberly	Harris	2041-03-25	+0761749605	B-	[n]
2	200041	Jaclyn	Smith	2061-08-13	+4053020325	O+	[n]
3	200040	Kathryn	Hunter	2034-06-02	+3653756430	AB-	[n]
4	200039	Thomas	Watkins	2008-09-14	+2019924443	O-	[n]
5	200038	Lauren	Cook	2022-01-11	+3308753432	B+	[n]
6	200037	Robin	Lang	2022-02-24	+9533242165	A+	[n]
7	200036	Andrew	Ford	2042-06-28	+4668320109	B-	[n]
8	200035	Robert	Dawson	2029-03-18	+4661353018	B+	[n]
9	200034	Andrew	Murray	2016-07-14	+9564885558	B+	[n]
10	200033	Cathy	Brennan	2027-11-22	+8983323175	B-	[n]
11	200032	Robert	Gross	2031-03-27	+7993667818	O-	[n]
12	200031	Ryan	Moore	2036-06-07	+7822002702	O-	[n]
13	200030	Stephen	Smith	2033-03-27	+5458714916	AB-	[n]
14	200029	Natalie	Webb	2041-07-01	+1001129225	AB-	[n]
15	200028	Jim	Farmer	2042-04-13	+5307438877	B+	[n]
16	200027	Jill	Dougherty	2040-05-29	+5066880141	AB+	[n]
17	200026	Lorraine	West	2030-11-03	+4222766701	B-	[n]

При генерації імен, фамілій і т.д. використовувалася генерація з бібліотеки faker, але в model.py можна вимкнути використання бібліотеки, й тоді назви, імена та інше будуть генеруватися повністю SQL запитом.

1

2

3

```
INSERT INTO "BloodBag" ( "BloodType", "StorageTemperature") VALUES
((array['A+', 'O+', 'B+', 'AB+', 'A-', 'O-', 'B-', 'AB-'])[floor(random()*8+1)], floor(random()*3 + 18))
RETURNING *
```

Data Output

Messages

Notifications

	BagID [PK] integer	BloodType text	StorageTemperature smallint
1	1032	AB+	19

1

2

3

```
INSERT INTO "BloodBag-BloodDonation" ("BloodDonationID", "BloodBagID") VALUES (
(SELECT "DonationID" FROM "BloodDonation" ORDER BY RANDOM() LIMIT 1),
(SELECT "BagID" FROM "BloodBag" ORDER BY RANDOM() LIMIT 1)) RETURNING "BloodDonationID", "BloodBagID"
```

Data Output

Messages

Notifications

	BloodDonationID [PK] integer	BloodBagID [PK] integer
1	6347	316

Query

Query History

1

2

3

4

5

6

7

8

```
INSERT INTO "BloodBank" ("Name", "Location", "ContactNumber", "TotalDonations",
"BloodDonationID") VALUES(
(array['HelpAnother', 'BeDonor', 'Back4Blood', 'Bloody', 'MealsForBlood',
'MeatHook', 'TheCakeisLie', 'Bloodborn'])[floor(random()*8+1)],
(array['Ukraine', 'USA', 'France', 'Germany', 'UK', 'Poland', 'Canada', 'India'])[floor(random()*8+1)],
to_char(random() * 10000000000, 'FM'+''000''000''0000'),
floor(random()*1000+10),
(SELECT "DonationID" FROM "BloodDonation" ORDER BY RANDOM() LIMIT 1)) RETURNING *
```

Data Output

Messages

Notifications

	BloodBankID [PK] integer	Name text	Location text	ContactNumber text	TotalDonations bigint	BloodDonationID integer
1	138	BeDonor	Poland	+1917459902	275	5095

Query Query History

```

1 INSERT INTO "BloodDonation" ("DonationDate", "DonationTime","DonorID",
2 "DonationStatus") VALUES(
3 date '2077-01-01' + (random() * (date '2000-11-11' - date '2077-01-01'))::int,
4 timestamp '2077-01-01' + random() * (timestamp '2000-11-11' - timestamp '2077-01-01'),
5 (SELECT "DonorID" FROM "Donor" ORDER BY RANDOM() LIMIT 1),
6 (array['Done', 'Planned'])[floor(random()*1+1)] RETURNING *

```

Data Output Messages Notifications

	DonationID [PK] integer	DonationDate date	DonationTime timestamp without time zone	DonorID integer	DonationStatus text
1	10042	2026-01-15	2027-12-20 15:00:16.411497	282618	Done

Query Query History

```

1 INSERT INTO "Recipient" ("FirstName", "LastName", "DateOfBirth",
2 "ContactNumber", "BloodTypeNeeded", "BloodBagID") VALUES (
3 (array['Ivan', 'Bora', 'Jorik','Olesia','Patrick', 'John', 'Steven', 'Nastya'])[floor(random()*8+1)],
4 (array['Terentiev', 'Zorev', 'Smith','Miller','Williams', 'Davis', 'Steven', 'Garcia'])[floor(random()*8+1)],
5 date '2077-01-01' + (random() * (date '2000-11-11' - date '2077-01-01'))::int,
6 to_char(random() * 10000000000, 'FM'+'000'000'0000'),
7 (array['A+', 'O+', 'B+', 'AB+', 'A-', 'O-', 'B-', 'AB-'])[floor(random()*8+1)],NULL) RETURNING *

```

Data Output Messages Notifications

	RecipientID [PK] integer	FirstName text	LastName text	DateOfBirth date	ContactNumber text	BloodTypeNeeded text	BloodBagID integer
1	200044	Ivan	Miller	2053-11-03	+8212775868	AB+	[null]

У випадку з Recipient не генерується BloodBagID, так як реципієнт може бути доданий до таблиці, коли відсутні пакети крові. Всі інші foreign keys генеруються як показано на скріншотах.

Завдання №3

Input params:
1) WHERE BloodBag.BloodType
Input value: >? A+
2) OR BloodBag.BloodType
Input value: >? B-
3) AND DonationStatus
Input value: >? Done

	1431		2075-05-30		Done	
	9047		2069-05-04		Done	
	3423		2025-12-02		Done	
	1611		2043-02-04		Done	
	5055		2063-11-19		Done	
	9085		2041-07-08		Done	
	5864		2049-03-13		Done	
	6863		2058-01-22		Done	
	9753		2043-10-18		Done	
	8040		2047-08-18		Done	
+-----+-----+-----+-----+						
Execution time: -0.016715049743652344						

1	SELECT	"DonationID", "DonationDate", "DonationStatus"	FROM	"BloodDonation"
2	INNER JOIN	"BloodBag-BloodDonation"	ON	"BloodDonationID"="DonationID"
3	INNER JOIN	"BloodBag"	ON	
4	"BloodBag-BloodDonation".	"BloodBagID"	=	"BloodBag"."BagID"
5	WHERE	("BloodBag"."BloodType"	=	'A+'
6	OR	"BloodBag"."BloodType"	=	'B-')
7	AND	"BloodDonation"."DonationStatus"	=	'Done'

Data Output

Messages

Notifications

	DonationID [PK] integer	DonationDate date	DonationStatus text
93	8056	2056-03-24	Done
94	9866	2052-09-02	Done
95	3397	2049-12-07	Done
96	3366	2010-09-15	Done
97	2018	2022-10-05	Done
98	7760	2042-12-11	Done
99	9467	2005-01-22	Done
100	1716	2021-06-16	Done
101	4617	2025-01-09	Done
102	6227	2013-09-17	Done
103	6304	2039-06-08	Done
104	1431	2075-05-30	Done
105	9047	2069-05-04	Done
106	3423	2025-12-02	Done
107	1611	2043-02-04	Done
108	5055	2063-11-19	Done
109	9085	2041-07-08	Done
110	5864	2049-03-13	Done
111	6863	2058-01-22	Done
112	9753	2043-10-18	Done
113	8040	2047-08-18	Done

Total rows: 113 of 113

Query complete 00:00:00.080

```

Input params:
1) WHERE BloodDonation.DonationStatus
Input value: >? Done
2) AND Donor.FirstName
Input value: >? Edward
3) AND Donor.ContactNumber
Input value: >? +3802557869
+-----+-----+-----+
| DonorID | FirstName | LastName |
+-----+-----+-----+
| 205153 | Edward | Colon |
+-----+-----+-----+
Execution time: -0.0220489501953125













```

```

1  SELECT "Donor"."DonorID", "FirstName", "LastName" FROM "Donor"
2  INNER JOIN "BloodDonation"
3  ON "BloodDonation"."DonorID" = "Donor"."DonorID"
4  WHERE "DonationStatus" = 'Done'
5  AND "FirstName" = 'Edward'
6  AND "ContactNumber" = '+3802557869'

```

Data Output Messages Notifications

								
	DonorID [PK] integer 	FirstName text 	LastName text 					
1	205153	Edward	Colon					


```

Input params:
1) WHERE BloodBag.BloodType
Input value: >? A-
2) AND Recipient.FirstName
Input value: >? Edward
3) AND Recipient.LastName
Input value: >? Reese
+-----+-----+-----+
| RecipientID | FirstName | LastName |
+-----+-----+-----+
|    103045   |   Edward |   Reese   |
+-----+-----+-----+
Execution time: -0.0017066001892089844

```

```

1  SELECT "RecipientID", "FirstName", "LastName" FROM "Recipient"
2  INNER JOIN "BloodBag" ON "Recipient"."BloodBagID" = "BloodBag"."BagID"
3  WHERE "BloodBag"."BloodType" = 'A-'
4  AND ("Recipient"."FirstName" = 'Edward'
5  AND "Recipient"."LastName" = 'Reese') GROUP BY "RecipientID"
6

```

Data Output	Messages	Notifications
<div> <div> <div>≡</div> <div>+</div> </div> <div> <div>📄</div> <div>▼</div> </div> <div> <div>📋</div> <div>▼</div> </div> <div> <div>🗑️</div> </div> <div> <div>🗄️</div> </div> <div> <div>⬇️</div> </div> <div> <div>📈</div> </div> </div>		
	RecipientID [PK] integer	<div> <div>FirstName</div> <div>text </div> </div> <div> <div>LastName</div> <div>text </div> </div>
1	103045	Edward Reese

Завдання №4



Використаний шаблон проекту MVC

Опис model.py

Тут створений клас Model, функції генерації рандомізованих даних(точніше імен, назв і т.п.) за допомогою бібліотеки faker, та функція перевірки введених користувачем значень, а саме перевірка на відповідність до типу даних і зберігання логіки моделі, як наприклад максимальна та мінімальна температура чи типи крові.

model.py це:

bool use_faker: Вмикає чи вимикає рандомізовані назви, імена та інші з бібліотеки faker

def verify_value: Перевіряє відповідність до типу даних, та зберігання логіки моделі

def generate_first_name: Генерує випадкове ім'я

def generate_last_name: Генерує випадкову фамілію

def generate_random_company: Генерує випадкову назву компанії

def generate_location: Генерує випадкове місцезнаходження

class Model це:

tables = dict(): Словник назв таблиць та їх стовпчики

tables_for_search = dict(): Словник пошукових запитів та назв таблиць

tables_for_rand = dict(): Словник запитів для рандомізованих даних

def __init__: Тут відбувається ініціалізація та підключення до бази даних

def execute: Функція, що робить запит до бази даних, повертає курсор для обробки, наприклад виводу таблиці

def get_table: Функція, що повертає вивід всіх стовпчиків обраної таблиці

def get_params: Функція, що повертає список стовпчиків обраної таблиці

def get_params_for_search: Функція, що повертає список констант для пошукового запиту

def search: Функція, що виконує пошуковий запит, повертає таблицю та час виконання пошуку

def get_typeof: Функція, що виконує запит на отримання типу даних обраного стовпчика з таблиці

def add_table: Функція, що додає рядок заповнений вручну користувачем

def check_id: Функція, що виконує запит на перевірку зайнятості ID

def delete_table: Функція, що виконує запит на видалення рядку з таблиці

def edit_table: Функція, що виконує запит на редагування стовпчика рядка з таблиці

def randomize_table: Функція, що виконує n-кількість запитів до бази даних, та генерує рядки обраної таблиці, якщо використовується faker, використовується локальна версія tables_for_rand для генерації назв.

Лістинг:

main.py

```
from controller import Controller
```

```
connection_settings = {  
    'dbname': 'postgres',  
    'user': 'postgres',  
    'password': 't3',  
    'host': 'localhost',  
    'port': 5432  
}
```

```
if __name__ == '__main__': # Startup controller  
    controller = Controller(connection_settings)  
    controller.run()
```

controller.py

```
import psycpg2
```

```
from model import Model
```

```
from view import View
```

```
class Controller:
```

```
    def __init__(self, connection_settings): # Startup for model and view
```

```
        self.view = View()
```

```
    try:
```

```
        self.model = Model(connection_settings)
```

```
    except psycpg2.Error:
```

```
        self.view.show_connection_error()
```

```
        exit(-1)
```

```
def run(self): # Menu
```

```
    menu = {
```

```
        1: self.view_table,
```

```
        2: self.view_all_tables,
```

```
        3: self.add_table,
```

```
        4: self.delete_table,
```

```
        5: self.delete_table,
```

```
        6: self.randomize_table,
```

```
        7: self.search
```

```
    }
```

```
    while True:
```

```
        choice = self.view.show_menu()
```

```
        if choice == 8:
```

```
            break
```

```
elif choice in menu:
```

```
    menu[choice]()
```

```
def view_table(self):
```

```
    try:
```

```
        selected = self.view.show_table_menu(self.model.tables)
```

```
        table = self.model.get_table(selected)
```

```
        self.view.show_table(table)
```

```
    except psycopg2.Error as error:
```

```
        self.view.show_sql_error(error)
```

```
def view_all_tables(self):
```

```
    try:
```

```
        for i in self.model.tables:
```

```
            table = self.model.get_table(i)
```

```
            self.view.show_msg(self.model.tables[i])
```

```
            self.view.show_table(table)
```

```
    except psycopg2.Error as error:
```

```
        self.view.show_sql_error(error)
```

```
def add_table(self):
```

```
    try:
```

```
        selected = self.view.show_table_menu(self.model.tables)
```

```

table = self.model.get_table(selected)

self.view.show_table(table)

needed_params = self.model.get_params(selected)

entered_params = self.view.show_params_menu(needed_params)

Verification = self.model.add_table(selected, entered_params)

if not Verification:

    self.view.show_sanity_error()

except psycopg2.Error as error:

    self.view.show_sql_error(error)

def delete_table(self):

    try:

        selected = self.view.show_table_menu(self.model.tables)

        table = self.model.get_table(selected)

        self.view.show_table(table)

        self.view.show_msg("Select id to delete: ")

        id_to_delete = self.view.get_id()

        Verification = self.model.delete_table(selected, id_to_delete)

        if not Verification:

            self.view.show_sanity_error()

    except psycopg2.Error as error:

        self.view.show_sql_error(error)

```

```

def edit_table(self):

    try:

        selected = self.view.show_table_menu(self.model.tables)

        table = self.model.get_table(selected)

        self.view.show_table(table)

        self.view.show_msg("Select id to edit: ")

        id_to_edit = self.view.get_id()

        available_params = self.model.get_params(selected)

        selected_param =
self.view.show_params_menu_selection(available_params)

        selected_param_value =
self.view.get_param(available_params[selected_param])

        Verification = self.model.edit_table(selected, id_to_edit,
selected_param, selected_param_value)

        if not Verification:

            self.view.show_sanity_error()

    except psycopg2.Error as error:

        self.view.show_sql_error(error)


def randomize_table(self):

    try:

        selected = self.view.show_table_menu(self.model.tables)

        table = self.model.get_table(selected)

```



```

        self.view.show_table(table)

        random_count = self.view.show_random_menu()

        self.model.randomize_table(selected, random_count)

    except psycopg2.Error as error:

        self.view.show_sql_error(error)


def search(self):

    try:

        selected_search_query =
self.view.show_table_menu(self.model.tables_for_search)

        available_params =
self.model.get_params_for_search(selected_search_query)

        entered_params = self.view.show_params_menu(available_params)

        table, execution_time = self.model.search(selected_search_query,
entered_params)

        self.view.show_table(table)

        self.view.show_execution_time(execution_time)

    except psycopg2.Error as error:

        self.view.show_sql_error(error)

```

view.py

```

from prettytable import from_db_cursor

```

```

class View(object):

    @staticmethod
    def show_table_menu(tables):

        num = -1

        while not (list(tables)[0] <= num <= list(tables)[-1]):

            print("Select table:")

            for table in tables:

                print(str(table) + ") " + tables[table][1])

            inp = None

            while inp is None or inp == "":

                inp = input("Input number: ")

            num = int(inp)

        return num

    @staticmethod
    def show_msg(message):

        print(message)

    @staticmethod
    def show_menu():

        num = -1

```

```
while not (num > 0 & num < 9):

    print("\nMenu: ")

    print("1) View table")

    print("2) View all tables")

    print("3) Add to table")

    print("4) Delete from table")

    print("5) Edit table")

    print("6) Randomize table")

    print("7) Search")

    print("8) Exit")

    inp = None

    while inp is None or inp == "":

        inp = input("Input number: ")

        num = int(inp)

    return num
```

```
@staticmethod
```

```
def show_table(table):
```

```
    print(from_db_cursor(table))
```

```
@staticmethod
```

```
def show_params_menu(params):
```

```
    print("\n Input params: ")
```

```
entered_params = {}

for param in params:

    print(str(param) + ") " + params[param])

    inp = None

    while inp is None or inp == "":

        inp = input("Input value: ")

    entered_params[param] = inp


return entered_params
```

```
@staticmethod
```

```
def get_id():
```

```
    inp = None

    while inp is None or inp == "":

        inp = input("Enter id: ")

    return int(inp)
```

```
@staticmethod
```

```
def show_params_menu_selection(params):
```

```
    print("\n Select param: ")

    num = -1

    while not (list(params)[0] <= num <= list(params)[-1]):

        for param in params:
```

```
print(str(param) + ") " + params[param])
```

```
inp = None
```

```
while inp is None or inp == ":
```

```
inp = input("Enter number: ")
```

```
num = int(inp)
```

```
return num
```

```
@staticmethod
```

```
def get_param(param):
```

```
inp = None
```

```
while inp is None or inp == ":
```

```
inp = input("Input value for {}: ".format(param))
```

```
return inp
```

```
@staticmethod
```

```
def show_random_menu():
```

```
inp = None
```

```
while inp is None or inp == ":
```

```
inp = input("Input count of random elements: ")
```

```
return int(inp)
```

```
@staticmethod
```

```
def show_execution_time(time):  
    print("Execution time: {}".format(time))
```

```
@staticmethod
```

```
def show_sql_error(error):  
    print("Got error from server:\n", error)
```

```
@staticmethod
```

```
def show_sanity_error():  
    print("Entered value(s) is(are) not correct")
```

```
@staticmethod
```

```
def show_connection_error():  
    print("Connection was not established")
```

model.py

```
import re
```

```
import time
```

```
from datetime import datetime
```

```
import faker
```

```
from psycopg2 import connect
```

```
import phonenumbers
```

```
use_faker = True
```

```
faker.Faker.seed(time.time())
```

```
fake = faker.Faker()
```

```
def verify_value(selected_table, selected_param, entered_param, typeof) ->
bool:
```

```
    format_for_date = "%Y-%m-%d"
```

```
    format_for_timestamp = "%Y-%m-%d %H-%M-%S"
```

```
    if (((selected_table == 1 and selected_param == 2) or (selected_table == 5
and selected_param == 6) or
```

```
        (selected_table == 6 and selected_param == 6)) and
```

```
        entered_param not in ['A+', 'O+', 'B+', 'AB+', 'A-', 'O-', 'B-', 'AB-']):
```

```
        # BloodBag.BloodType, Recipient.BloodTypeNeeded, Donor.BloodType
```

```
        return False
```

```
    if ((selected_table == 1 and selected_param == 3) and
```

```
        not (17 < int(entered_param) < 24)): # BloodBag.StorageTemperature
```

```
        return False
```

```
    if typeof == 'date':
```

```
        try:
```

```
            check_date = bool(datetime.strptime(entered_param,
format_for_date))
```

```
        except ValueError:
```

```

        check_date = False

    return check_date

if typeof == 'timestamp without time zone':

    try:

        check_date = bool(datetime.strptime(entered_param,
format_for_timestamp))

    except ValueError:

        check_date = False

    return check_date

if (selected_table == 3 and selected_param == 4) or (selected_table == 5 and
selected_param == 5) or (selected_table == 6
and selected_param == 5):

    # ContactNumber

    phone_number = phonenumber.parse(entered_param)

    if not phonenumber.is_possible_number(phone_number):

        return False

    if (typeof == 'integer' or typeof == 'smallint' or typeof == 'bigint') and
instance(entered_param, str):

        if not (entered_param.isnumeric()):

            return False

    if typeof == 'text' and not (selected_table == 3 and selected_param == 3): #
Address can contain all symbols

        if not re.search("^[A-Z][a-z]", entered_param):

```



```
return False
```

```
return True
```

```
def generate_first_name() -> str:
```

```
    if use_faker:
```

```
        res = "" + fake.first_name() + "" + ","
```

```
    else:
```

```
        res = (f"({array['Ivan', 'Bora', 'Jorik', 'Olesia', "
```

```
                f"'Patrick', 'John', 'Steven', 'Nastya'])[floor(random()*8+1)],")
```

```
    return res
```

```
def generate_last_name() -> str:
```

```
    if use_faker:
```

```
        res = "" + fake.last_name() + "" + ","
```

```
    else:
```

```
        res = (f"(array['Terentiev', 'Zorev', 'Smith', 'Miller', "
```

```
                f"'Williams', 'Davis', 'Steven', 'Garcia'])[floor(random()*8+1)],")
```

```
    return res
```

```
def generate_random_company() -> str:
```

```

if use_faker:

    res = "" + fake.company() + "" + ","

else:

    res = (f"(array['HelpAnother', 'BeDonor', 'Back4Blood','Bloody',
'MealsForBlood', "

f"'MeatHook', 'TheCakeislie','Bloodborn'])[floor(random()*8+1)],")

return res

```

```

def generate_location() -> str:

```

```

if use_faker:

    res = "" + fake.country() + ", " + fake.city() + ", " +
fake.street_address() + "" + ", "

else:

    res = f"(array['Ukraine', 'USA', 'France','Germany', 'UK', 'Poland',
'Canada', 'India'])[floor(random()*8+1)],"

return res

```

```

class Model:

```

```

    tables = {

        1: {1: 'BloodBag',

            2: {1: 'BagID', 2: 'BloodType', 3: 'StorageTemperature'}}},

```

```

2: {1: 'BloodBag-BloodDonation',
    2: {1: 'BloodDonationID', 2: 'BloodBagID'}},
3: {1: 'BloodBank',
    2: {1: 'BloodBankID', 2: 'Name', 3: 'Location', 4: 'ContactNumber', 5:
'TotalDonations',
    6: 'BloodDonationID'}},
4: {1: 'BloodDonation',
    2: {1: 'DonationID', 2: 'DonationDate', 3: 'DonationTime', 4:
'DonorID', 5: 'DonationStatus'}},
5: {1: 'Donor',
    2: {1: 'DonorID', 2: 'FirstName', 3: 'LastName', 4: 'DateOfBirth',
    5: 'ContactNumber', 6: 'BloodType'}},
6: {1: 'Recipient',
    2: {1: 'RecipientID', 2: 'FirstName', 3: 'LastName', 4: 'DateOfBirth',
    5: 'ContactNumber', 6: 'BloodTypeNeeded', 7: 'BloodBagID'}}
}

```

```

tables_for_search = {
    1: {1: 'BloodDonation',
        2: {1: 'WHERE BloodBag.BloodType', 2: 'OR BloodBag.BloodType',
3: 'AND DonationStatus'},
        3: 'SELECT "DonationID", "DonationDate", "DonationStatus"
FROM "BloodDonation" '
        'INNER JOIN "BloodBag-BloodDonation" ON
"BloodDonationID"="DonationID" '

```

```

'INNER JOIN "BloodBag" ON '

'"BloodBag-BloodDonation"."BloodBagID" = "BloodBag"."BagID"

,

f'WHERE ("BloodBag"."BloodType" = %s '

f'OR "BloodBag"."BloodType" = %s) AND '

f'"BloodDonation"."DonationStatus" = %s'}},

2: {1: 'Donor',

2: {1: 'WHERE BloodDonation.DonationStatus', 2: 'AND

Donor.FirstName', 3: 'AND Donor.ContactNumber'}},

3: 'SELECT "Donor"."DonorID", "Donor"."FirstName", '

'"Donor"."LastName" FROM "Donor" INNER JOIN '

'"BloodDonation" ON "Donor"."DonorID" =

"BloodDonation"."DonorID" '

f'WHERE "BloodDonation"."DonationStatus" = %s AND '

f('"Donor"."FirstName" = %s AND '

f'"Donor"."ContactNumber" = %s) '

'GROUP BY "Donor"."DonorID"}},

3: {1: 'Recipient',

2: {1: 'WHERE BloodBag.BloodType', 2: 'AND Recipient.FirstName',

3: 'AND Recipient.LastName'}},

3: 'SELECT "RecipientID", "FirstName", "LastName" FROM

"Recipient" '

'INNER JOIN "BloodBag" ON "Recipient"."BloodBagID" =

"BloodBag"."BagID" '

```

```

f'WHERE "BloodBag"."BloodType" = %s '

f'AND ("Recipient"."FirstName" = %s '

f'AND "Recipient"."LastName" = %s) GROUP BY
"RecipientID"}},

}

tables_for_rand = {

1: f'INSERT INTO \"BloodBag\" ( \"BloodType\",
\"StorageTemperature\") VALUES \"

f'((array['A+', 'O+', 'B+', 'AB+', 'A-', 'O-', 'B-',
'AB-'])[floor(random()*8+1)], floor(random()*3 \"

f\"+ 18))\",

2: f'INSERT INTO \"BloodBag-BloodDonation\" (\"BloodDonationID\",
\"BloodBagID\") VALUES (\"

f'(SELECT \"DonationID\" FROM \"BloodDonation\" ORDER BY
RANDOM() LIMIT 1),\"

f'(SELECT \"BagID\" FROM \"BloodBag\" ORDER BY RANDOM()
LIMIT 1))\",

3: f'INSERT INTO \"BloodBank\" (\"Name\", \"Location\",
\"ContactNumber\", \"TotalDonations\", \"

f\" \"BloodDonationID\") VALUES(\" +

generate_random_company() +

generate_location() +

f'to_char(random() * 10000000000, 'FM\"+\"000\"\"000\"\"0000'),\"

f'floor(random()*1000+10),\"

```

```
f"(SELECT \"DonationID\" FROM \"BloodDonation\" ORDER BY  
RANDOM() LIMIT 1))",
```

```
4: f"INSERT INTO \"BloodDonation\" (\"DonationDate\",  
\"DonationTime\", \"DonorID\", "
```

```
f\"\"DonationStatus\") VALUES("
```

```
f"date '2077-01-01' + (random() * (date '2000-11-11' - date  
'2077-01-01'))::int,"
```

```
f"timestamp '2077-01-01' + random() * (timestamp '2000-11-11' -  
timestamp '2077-01-01'),"
```

```
f"(SELECT \"DonorID\" FROM \"Donor\" ORDER BY RANDOM()  
LIMIT 1),"
```

```
f"(array['Done', 'Planned'])[floor(random()*1+1))",
```

```
5: f"INSERT INTO \"Donor\" (\"FirstName\", \"LastName\",  
\"DateOfBirth\", \"ContactNumber\", "
```

```
f\"\"BloodType\")"
```

```
f"VALUES(" +
```

```
generate_first_name() +
```

```
generate_last_name() +
```

```
f"date '2077-01-01' + (random() * (date '2000-11-11' - date  
'2077-01-01'))::int,"
```

```
f"to_char(random() * 10000000000, 'FM\"'+\"000\"\\\"000\"\\\"0000'),"
```

```
f"(array['A+', 'O+', 'B+', 'AB+', 'A-', 'O-', 'B-'  
'AB-'])[floor(random()*8+1))",
```

```
6: f"INSERT INTO \"Recipient\" (\"FirstName\", \"LastName\",  
\"DateOfBirth\", "
```

```

        f" \"ContactNumber\", \"BloodTypeNeeded\", \"BloodBagID\")
VALUES (" +
        generate_first_name() +
        generate_last_name() +
        f"date '2077-01-01' + (random() * (date '2000-11-11' - date
'2077-01-01'))::int,"
        f"to_char(random() * 10000000000, 'FM\"'+\"000\"\\\"000\"\\\"0000'),"
        f"(array['A+', 'O+', 'B+', 'AB+', 'A-', 'O-', 'B-',
'AB-'])[floor(random()*8+1)],"
        f"NULL)")
    }

```

```

def __init__(self, connection_settings):
    self.connection = connect(
        dbname=connection_settings['dbname'],
        user=connection_settings['user'],
        password=connection_settings['password'],
        host=connection_settings['host'],
        port=connection_settings['port']
    )

```

```

def execute(self, execute_string, params=None):
    cursor = self.connection.cursor()

```

```
cursor.execute(execute_string, params)
```

```
self.connection.commit()
```

```
return cursor
```

```
def get_table(self, selected_table):
```

```
    return self.execute(f'SELECT * FROM  
"{self.tables[selected_table][1]}"')
```

```
def get_params(self, selected_table):
```

```
    return self.tables[selected_table][2]
```

```
def get_params_for_search(self, selected_table):
```

```
    return self.tables_for_search[selected_table][2]
```

```
def search(self, selected_table, entered_params):
```

```
    start = time.time()
```

```
    res = self.execute(self.tables_for_search[selected_table][3],  
                       (str(entered_params[1]), str(entered_params[2]),  
str(entered_params[3])))
```

```
    end = time.time()
```

```
    return res, (end - start)
```

```
def get_typeof(self, selected_table, selected_param):
```



```

        cursor = self.execute(f'SELECT data_type FROM
information_schema.columns WHERE table_schema = \'public\' AND '
                               f'table_name = \'{self.tables[selected_table][1]}\' AND '
                               f'column_name =
\'{self.tables[selected_table][2][selected_param]}\'')

        from_cursor = cursor.fetchall()

        return from_cursor[0][0]

```

```

def edit_param_to_real_string(self, selected_table, selected_param,
entered_param):

    typeof = self.get_typeof(selected_table, selected_param)

    if typeof == 'text' or typeof == 'date' or typeof == 'timestamp without
time zone':

        return "" + entered_param + ""

    return entered_param

```

```

def add_table(self, selected_table, entered_params) -> bool:

    Verification = True

    params_string = ', '.join(f''{str(self.tables[selected_table][2][x])}'' for x
in self.tables[selected_table][2])

    for i in entered_params:

        entered_params[i] = self.edit_param_to_real_string(selected_table, i,
entered_params[i])

```

```

        Verification = verify_value(selected_table, i, entered_params[i],
self.get_typeof(selected_table, i))

        entered_params_string = ', '.join(str(entered_params[x]) for x in
entered_params)

        if Verification:

            self.execute(f'INSERT INTO public.{self.tables[selected_table][1]} ' '
+
                f'({params_string})'
                f'VALUES({entered_params_string})')

            return Verification

```

```

def check_id(self, selected_table, selected_id) -> bool:

    selected_id_real_num = verify_value(selected_table, 1, selected_id,
'integer')

    if selected_id_real_num:

        cursor = self.execute(f'SELECT FROM
public.{self.tables[selected_table][1]} WHERE'

            f' {self.tables[selected_table][2][1]} = {selected_id}')

        from_cursor = cursor.fetchall()

        if len(from_cursor) == 0:

            return False

        else:

            return False

    return True

```

```

def delete_table(self, selected_table, selected_id) -> bool:

    Verification = self.check_id(selected_table, selected_id)

    if Verification:

        self.execute(f'DELETE FROM public.{self.tables[selected_table][1]}'
,

                    f'WHERE "{self.tables[selected_table][2][1]}" = {selected_id}')

    return Verification


def edit_table(self, selected_table, selected_id, selected_param,
entered_param) -> bool:

    entered_param = self.edit_param_to_real_string(selected_table,
selected_param, entered_param)

    Verification = verify_value(selected_table, selected_param,
entered_param, self.get_typeof(selected_table,

                                selected_param))

    if Verification:

        Verification = self.check_id(selected_table, selected_id)

        if Verification:

            self.execute(f'UPDATE public.{self.tables[selected_table][1]} SET
,

                        f'{self.tables[selected_table][2][selected_param]}" =
{entered_param} WHERE '

                        f'{self.tables[selected_table][2][1]}" = {selected_id}')

```

return Verification

```
def randomize_table(self, selected_table, count):

    for _ in range(count):

        if use_faker:

            tables_for_rand = { # I did not find something like reinit, so it looks
like this for faker

                1: f"INSERT INTO \"{BloodBag}\" ( \"{BloodType}\",
                \"{StorageTemperature}\") VALUES "

                    f"((array['A+', 'O+', 'B+', 'AB+', 'A-', 'O-', 'B-',
                    'AB-'])[floor(random()*8+1)], "

                    f"floor(random()*3 + 18))",

                2: f"INSERT INTO \"{BloodBag-BloodDonation}\"
                (\"BloodDonationID\", \"BloodBagID\") VALUES ("

                    f"(SELECT \"DonationID\" FROM \"BloodDonation\" ORDER
                    BY RANDOM() LIMIT 1),"

                    f"(SELECT \"BagID\" FROM \"BloodBag\" ORDER BY
                    RANDOM() LIMIT 1))",

                3: f"INSERT INTO \"{BloodBank}\" (\"Name\", \"Location\",
                \"ContactNumber\", \"TotalDonations\", "

                    f"\"BloodDonationID\") VALUES(" +

                    generate_random_company() +

                    generate_location() +

                    f"to_char(random() * 10000000000,
                    'FM'+\"000\\\"000\\\"0000'),"
```

```

f"floor(random()*1000+10),"

f"(SELECT \"DonationID\" FROM \"BloodDonation\" ORDER
BY RANDOM() LIMIT 1))",

4: f"INSERT INTO \"BloodDonation\" (\"DonationDate\",
\"DonationTime\", \"DonorID\",

f\"\"DonationStatus\"") VALUES("

f"date '2077-01-01' + (random() * (date '2000-11-11' - date
'2077-01-01'))::int,"

f"timestamp '2077-01-01' + random() * (timestamp '2000-11-11'
- timestamp '2077-01-01'),"

f"(SELECT \"DonorID\" FROM \"Donor\" ORDER BY
RANDOM() LIMIT 1),"

f"(array['Done', 'Planned'])[floor(random()*1+1))",

5: f"INSERT INTO \"Donor\" (\"FirstName\", \"LastName\",
\"DateOfBirth\", \"ContactNumber\", "

f\"\"BloodType\"")

f"VALUES(" +

generate_first_name() +

generate_last_name() +

f"date '2077-01-01' + (random() * (date '2000-11-11' - date
'2077-01-01'))::int,"

f"to_char(random() * 10000000000,
'FM\"+\"000\"\"\"000\"\"\"0000'),"

f"(array['A+', 'O+', 'B+', 'AB+', 'A-', 'O-', 'B-',
'AB-'])[floor(random()*8+1))",

```

```

6: f"INSERT INTO \"Recipient\" (\"FirstName\", \"LastName\",
\"DateOfBirth\",
f\" \"ContactNumber\", \"BloodTypeNeeded\", \"BloodBagID\")
VALUES (" +
generate_first_name() +
generate_last_name() +
f"date '2077-01-01' + (random() * (date '2000-11-11' - date
'2077-01-01'))::int,"
f"to_char(random() * 10000000000,
'FM\"+\"000\"\"000\"\"0000'),"
f"(array['A+', 'O+', 'B+', 'AB+', 'A-', 'O-', 'B-',
'AB-'])[floor(random()*8+1)],"
f"NULL)"
}

self.execute(tables_for_rand[selected_table])

else:

self.execute(self.tables_for_rand[selected_table])

```