

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування та 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*»*

Виконав: студент ІІІ курсу

ФПМ групи КВ-93

Зеленяк І.О.

Перевірив:

Київ, 2021

Посилання на репозиторій <https://github.com/AmiriVANYCH/DataBase>

## Завдання на лабораторну роботу

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).

## Вступ

В даній лабораторній буде розроблятися програма керування парковкою з розділеними персональними місцями для кожної автівки, з використанням бази даних, що була розроблена в першій лабораторній роботі.

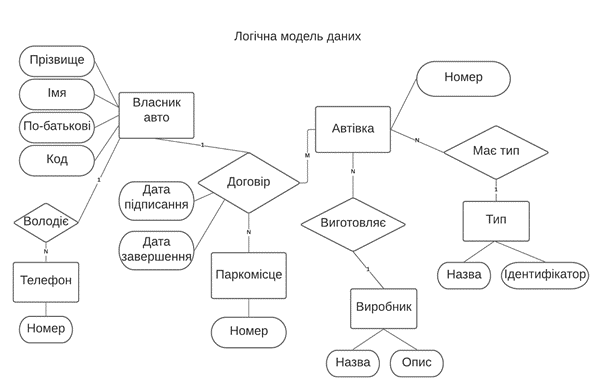
Парковка працює за таким приципом:

Для того, щоб особа користувалася парковкою необхідно підписати договір на окреме паркувальне місце.

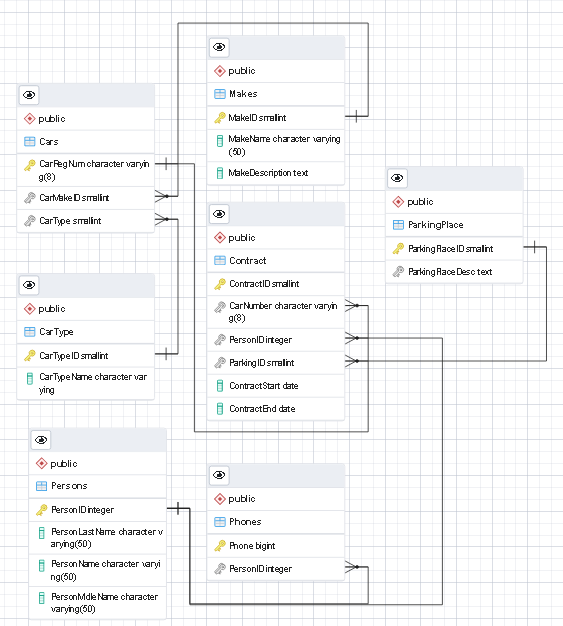
Вважаємо що в одного користувача може бути доступ до декількох автівок (декілька договорів).

Одному паркомісцю відповідає тільки одна машина яка може там стояти (за договором), якщо воно орендоване, інакше жодна машина не може там стояти.

## Логічна модель бази данних:



## Таблиці бази даних



### Опис стовбців їх данних І типів

|  |  |  |  |
| --- | --- | --- | --- |
| **Назва таблиці/поля** | **тип**  **даних** | **обов’язково** | **опис** |
| **Таблиця "Cars"** |  |  |  |
| *"CarRegNum"* | text | Так | Номер автівки |
| *"CarMakeID"* | smallint | Так | Ідентифікатор виробника автівки |
|  |  |  |  |
| **Таблиця "Makes"** |  |  |  |
| *"MakeID"* | smallint | Так | ідентифікатор марки авто |
| *"MakeName"* | text | Так | назва марки |
| *"MakeDescription"* | text | Ні | опис |
|  |  |  |  |
| **Таблиця "ParkingPlace"** |  |  |  |
| *"ParkingPlaceID"* | smallint | Так | ідентифікатор парковки |
| *"ParkingPlaceDesc"* | text | Ні | опис |
|  |  |  |  |
| **Таблиця "Persons"** |  |  |  |
| *"PersonID"* | bigint | Так | ідентифікатор персони |
| *"PersonLastName"* | text | Так | фамілія персони |
| *"PersonName"* | text | Так | ім’я персони |
| *"PersonMidleName"* | text | Ні | по батькові персони |
|  |  |  |  |
| **Таблиця "Phones"** |  |  |  |
| *"Pnone"* | bigint | Так | номер телефону |
| *"PersonID"* | bigint | Так | ідентифікатор персони |
|  |  |  |  |
| **Таблиця "CarType"** |  |  |  |
| *"CarTypeID"* | smallint | Так | ідентифікатор типу |
| *"CarTypeName"* | text | Так | назва типу |
|  |  |  |  |
| **Таблиця "Contract"** |  |  |  |
| *"ContractID"* | smallint | Так | номер договору |
| *"CarNumber"* | text | Так | Номер автівки |
| *"PersonID"* | bigint | Так | ідентифікатор персони |
| "ParkingID" | smallint | Так | ідентифікатор парковки |
| "ContractStart" | date | Так | Дата підписання договору |
| *"*ContractEnd*"* | date | Так | Дата завершення договору |

|  |  |  |
| --- | --- | --- |
| Меню програми | | |
| Додати договір | | Запускає поцедуру додаввання договору |
| Курувати виробниками авто | | Меню керування виробниками |
|  | Додати марку | Запускає поцедуру додаввання виробника |
|  | Видалити марку | Запускає поцедуру видалення виробника |
|  | Редагувати марку | Запускає поцедуру редагування виробника |
| Керувати типами авто | | Меню керування типами автомобілів |
|  | Додати тип авто | Запускає поцедуру додаввання типу авто |
|  | Видалити тип авто | Запускає поцедуру видалення типу авто |
|  | Редагувати тип авто | Запускає поцедуру редагування типу авто |
| Видалити контракт | | Меню видалення контрактів |
|  | За номером контракту | Запускає поцедуру видалення контракту за номером контракту |
|  | За номером автомобіля | Запускає поцедуру видалення контракту за номером автомобіля |
|  | За ідентифікаційним номером | Запускає поцедуру видалення контракту за ідентифікаційним номером |
| Створити пакетно багато контрактів | | Запускає процедуру пакетного генерування данних |
| Пошук контракту за декількома параметрами | | Запускає процедуру пошуку контракту за декількома параметрами |

## Робоче середовище

Робота виконана мовою Python

З використанням бібліотеки Psycopg

СУБД PostreSQL

Середовище роботи з СУБД pgAdmin 4

## Принцип роботи програми

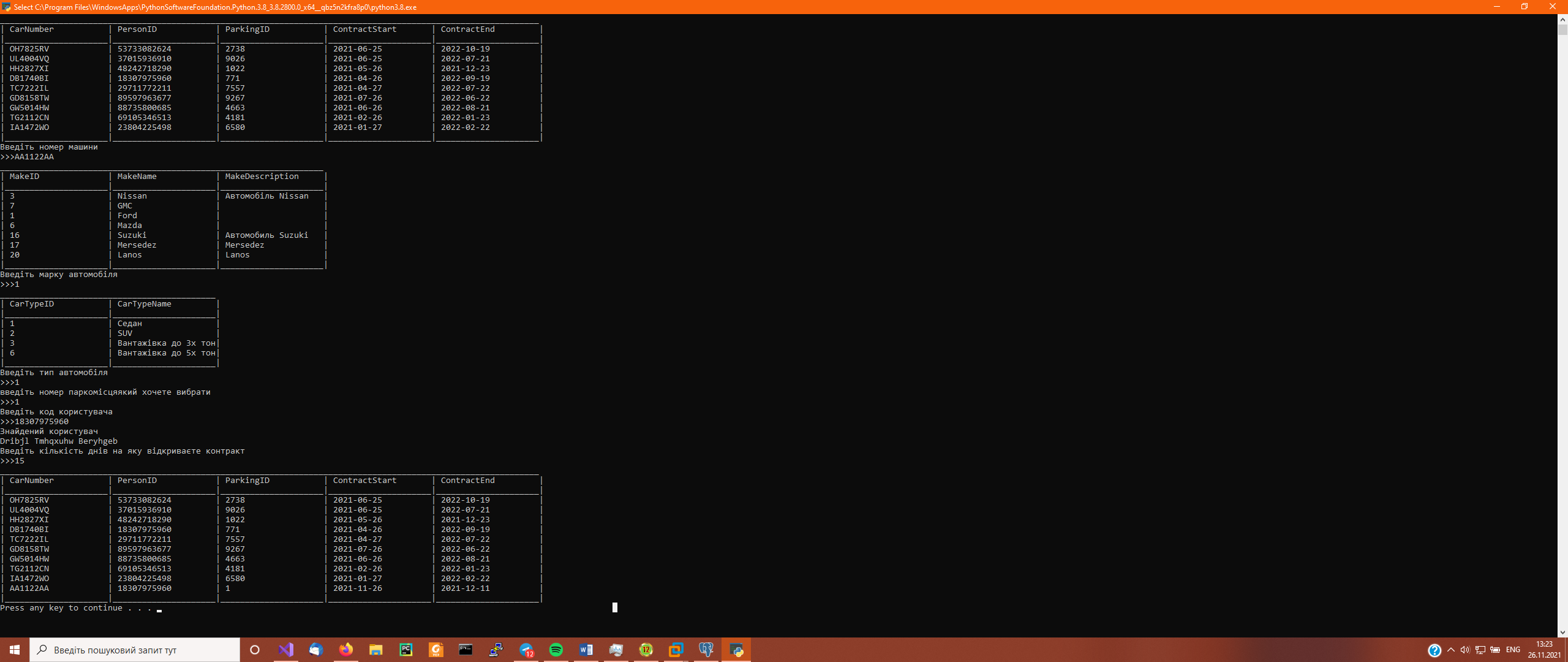
Програмний код виконаний згідно шаблону MVC (модель-подання-контролер).

Тобто в програмі використовуються три модулі:

* Model відповідає за роботу з базою данних
* Control відповідає за логічну частину виконаних дій програми
* View відповідає за взаємодію з користувачем

## Реалізація функції внесення змін до таблиць

Функції перегляду, внесення, редагування та вилучення даних у таблицях бази даних, реалізовані окремими функціями, що виконують по одній операції. Ці функції використовуються «контролером» для більш складної логіки. Результат роботи функцій репрезентує знімок екрану:



## Лістинг

1. **def** create\_Car (self,CarRegNum,CarMakeID,CarType):
2. items = self.model.create\_Car (CarRegNum,CarMakeID,CarType)
3. self.view.show\_items(items)
5. **def show\_Car (self,):**
6. items = self.model.read\_Car ()
7. self.view.show\_items(items)
9. **def** update\_Car (self,CarRegNum,CarMakeID,CarType):
10. **self.model.update\_Car (CarRegNum,CarMakeID,CarType)**
12. **def** del\_Car (self,CarRegNum):
13. self.model.del\_Car (CarRegNum)
14. **def** create\_Car (self ,CarRegNum,CarMakeID,CarType):
15. cursor = self.get\_cursor()
16. record = get\_Car\_by\_Reg\_num(CarRegNum)
17. **if** cursor.rowcount == 0:
18. **query = 'INSERT INTO public."Cars"("CarRegNum", "CarMakeID" , "CarType")' + "VALUES ('"+ str(CarRegNum)+"',"+ str(CarMakeID)+","+ str(CarType) +");"**
19. cursor.execute(query)
21. self.\_connection.commit()
22. **else**:
23. **makeid = record[0]**
25. **def** read\_Car (self ,):
26. cursor = self.get\_cursor()
27. query = 'SELECT "CarRegNum", "CarMakeID", "CarType" FROM public."Cars";'
28. **cursor.execute(query)**
29. self.\_connection.commit()
30. record = cursor.fetchall()
31. **return** record
33. **def update\_Car (self ,CarRegNum,CarMakeID,CarType):**
34. cursor = self.get\_cursor()
35. record = get\_Car\_by\_Reg\_num(CarRegNum)
36. **if** cursor.rowcount > 0:
37. query = 'UPDATE public."Cars" SET "CarType"='+"'"+str(CarType) +"'"+', "CarMakeID"='+"'"+str(CarMakeID) +"'"+ ' WHERE "CarRegNum" = '+ str(CarRegNum)+ ' ;'
38. **cursor.execute(query)**
39. self.\_connection.commit()

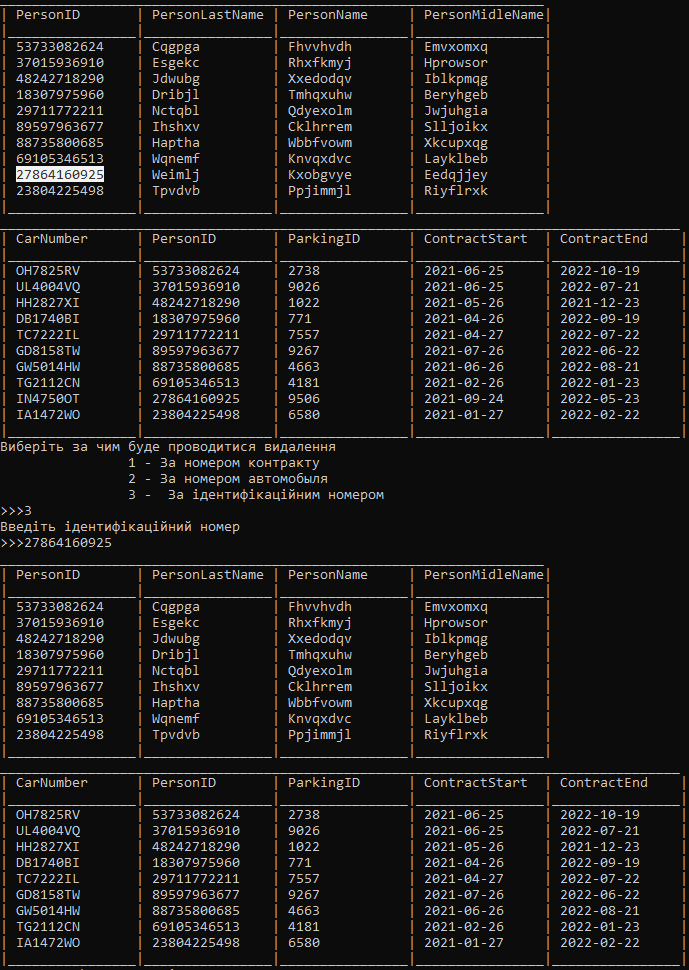
42. **def** del\_Car (self ,CarRegNum):
43. **cursor = self.get\_cursor()**
44. record = get\_Car\_by\_Reg\_num(CarRegNum)
45. **if** cursor.rowcount > 0:
46. query = 'DELETE FROM public."Cars" WHERE "CarRegNum"='+ str(CarRegNum)+ ';'
47. cursor.execute(query)
48. **self.\_connection.commit()**

## Видалення даних з пов’язаних таблиць

Як видно з логічної моделі бази данних, таблиця контрактів є ключовою, яка зєднує між собою автівки , персони та паркомісця.

При видаленні контракту втрачається звязок між машиною та персоною, а видалення персони чи машини на пряму не можливе так як значення використовуються в контрактах.

Для того щоб видалити контракт я створюю тимчасову таблицю та створюю в ній перелік записів які необхідно видалити. Використовуючи цю інформацію, видаляю контракти з основної таблиці контрактів, та видаляю данні з батьківських таблиць, якщо вони не повязані з іншими контрактами. На останньму кроці видаляю тимчасову таблицю.

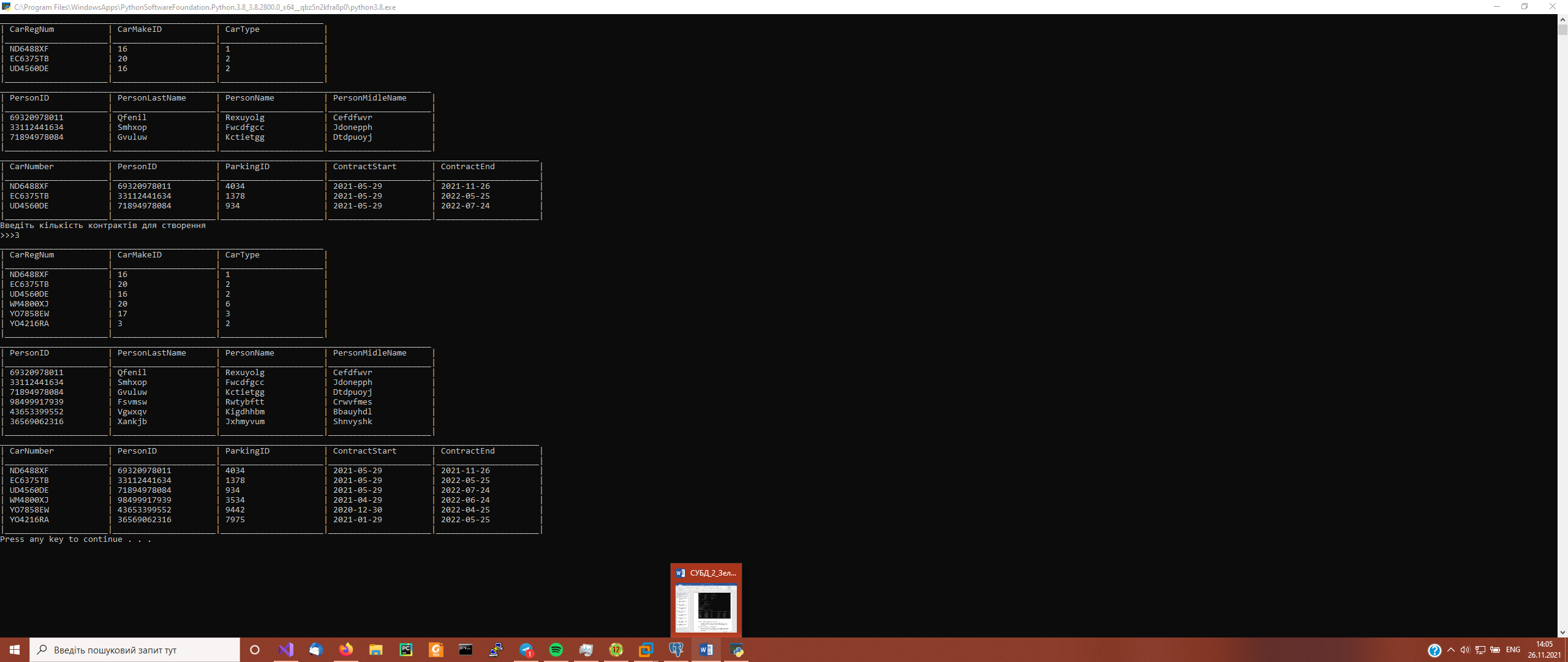


## Лістинг

1. **def** del\_contract\_by\_temp\_table\_CarNum(self,CarNum):
2. cursor = self.get\_cursor()
3. query ='''DROP TABLE IF EXISTS public."TMP\_Contract";
4. CREATE TABLE IF NOT EXISTS public."TMP\_Contract"("ContractID" smallint NOT NULL,"CarNumber" character varying(8) COLLATE pg\_catalog."default" NOT NULL,"PersonID" bigint NOT NULL,"ParkingID" smallint NOT NULL,"ContractStart" date NOT NULL,"ContractEnd" date NOT NULL);
5. insert into public."TMP\_Contract" ("ContractID","PersonID","CarNumber","ParkingID","ContractStart","ContractEnd") select "ContractID","PersonID","CarNumber","ParkingID","ContractStart","ContractEnd" from public."Contract" where "CarNumber" = /' '''+ str(CarNum)+ ''' /'' ;
6. delete from public."Contract" where "ContractID" in (select distinct "ContractID" from public."TMP\_Contract" );
7. delete from public."Cars" where "CarRegNum" = /' '''+ str(CarNum)+ ''' /';
8. delete from public."Persons" where "PersonID" in (select distinct "PersonID" from public."TMP\_Contract" ) and "PersonID" not in (select distinct "PersonID" from public."Contract" where "PersonID" in (select distinct "PersonID" from public."TMP\_Contract" ) );
9. DROP TABLE public."TMP\_Contract";'''
11. cursor.execute(query)
12. self.\_connection.commit()

## Автоматичне пакетне генерування «рандомізованих» даних у базі.

Для пакетного генерування данних я використовую збережену процедуру бази данних



## Лістинг

1. **def** random\_create (self, param):
2. cursor = self.get\_cursor()
3. query = "CALL public.generate\_data("+str(param)+");"
4. cursor.execute(query)
5. **self.\_connection.commit()**

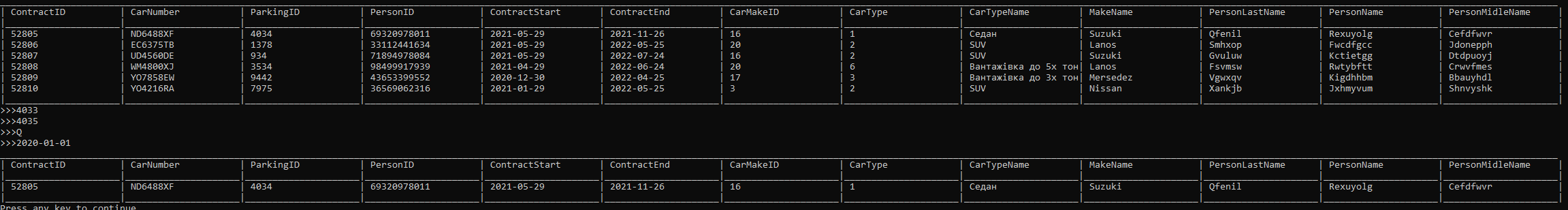
**Збережена процедура**

1. CREATE OR REPLACE PROCEDURE generate\_data(dataCount integer)
2. AS $$
3. 
4. DECLARE
5. i SMALLINT;
6. Persons RECORD;
7. CarNum text;
8. 
9. BEGIN
10. INSERT INTO public."Persons" ("PersonID","PersonLastName","PersonMidleName","PersonName")
11. SELECT floor(10000000000+random()\*89999999999)::bigint,
12. chr(floor(65 + random()\*25)::int) ||
13. chr(floor(97 + random()\*25)::int) || chr(floor(97 + random()\*25)::int) ||
14. chr(floor(97 + random()\*25)::int) || chr(floor(97 + random()\*25)::int) ||
15. chr(floor(97 + random()\*25)::int),
16. chr(floor(65 + random()\*25)::int) ||
17. chr(floor(97 + random()\*25)::int) || chr(floor(97 + random()\*25)::int) ||
18. chr(floor(97 + random()\*25)::int) || chr(floor(97 + random()\*25)::int) ||
19. chr(floor(97 + random()\*25)::int) || chr(floor(97 + random()\*25)::int) ||
20. chr(floor(97 + random()\*25)::int),
21. chr(floor(65 + random()\*25)::int) ||
22. chr(floor(97 + random()\*25)::int) || chr(floor(97 + random()\*25)::int) ||
23. chr(floor(97 + random()\*25)::int) || chr(floor(97 + random()\*25)::int) ||
24. chr(floor(97 + random()\*25)::int) || chr(floor(97 + random()\*25)::int) ||
25. chr(floor(97 + random()\*25)::int)
26. FROM generate\_series(1, dataCount);
27. 
28. FOR Persons IN
29. SELECT "PersonID" from public."Persons" where "PersonID" not in (SELECT DISTINCT "PersonID" from public."Contract")
30. LOOP
31. CarNum :=(chr(floor(65 + random()\*25)::int) ||
32. chr(floor(65 + random()\*25)::int) || floor( random()\*9)::int ||
33. floor( random()\*9)::int || floor( random()\*9)::int || floor( random()\*9)::int || chr(floor(65 + random()\*25)::int) ||
34. chr(floor(65 + random()\*25)::int));
35. INSERT INTO public."Cars" ("CarMakeID","CarRegNum","CarType")
36. SELECT (select "MakeID" from public."Makes" where "MakeID" >= (select floor(random()\* (SELECT MAX("MakeID") FROM public."Makes"))::int) limit 1) ,
37. CarNum,
38. (SELECT "CarTypeID" FROM public."CarType" ORDER BY random() LIMIT 1);
39. 
40. insert into public."Contract" ("CarNumber","ContractStart","ContractEnd","ParkingID","PersonID")
41. select
42. CarNum,
43. (SELECT CURRENT\_DATE - 1 -(30 \* floor( random()\*12)::int) ) ,
44. (SELECT CURRENT\_DATE + 30 \* floor( random()\*12)::int ),
45. floor( 1+ random()\*9999)::int,
46. Persons."PersonID";
47. END LOOP;
48. END;
49. $$ LANGUAGE plpgsql;

## Пошук інформації

Програма забезпечує реалізацію пошуку за декількома атрибутами з двох та більше сутностей одночасно. При ціьому для числових атрибутів – у рамках діапазону, для рядкових – як шаблон функції LIKE оператора SELECT SQL, для дат – у рамках діапазону дат.

Приклад результату роботи функції:



Лістинг функції

1. **def** select(self,parking\_start , parking\_end ,PersonPart, Date ):
2. cursor = self.get\_cursor()
3. query = '''select "ContractID", "CarNumber", "ParkingID", "Contract"."PersonID" as "PersonID", "ContractStart", "ContractEnd","CarMakeID","CarType", "CarTypeName", "MakeName", "PersonLastName", "PersonName", "PersonMidleName"
4. from public."Contract"
5. **INNER JOIN public."Cars" on public."Contract"."CarNumber" = public."Cars"."CarRegNum"**
6. INNER JOIN public."CarType" on public."Cars"."CarType" = public."CarType"."CarTypeID"
7. INNER JOIN public."Makes" on public."Makes"."MakeID" = public."Cars"."CarMakeID"
8. INNER JOIN public."Persons" on public."Persons"."PersonID" = public."Contract"."PersonID"
9. where "ContractID" > 1
10. **and "ParkingID" between '''+parking\_start +''' and '''+ parking\_end +'''**
11. and "PersonLastName" like '''+"'%"+ PersonPart +"%'"+'''
12. and "ContractEnd" > '''+"'"+ Date +"'"
14. cursor.execute(query)
15. **self.\_connection.commit()**
16. record = cursor.fetchone()
17. **return** record

## Текст файлу model

1. import mvc\_exceptions as mvc\_exc
2. import psycopg
3. from psycopg import Error
4. from psycopg.rows import dict\_row

7. class ModelBasic(object):
9. \_connection = None
10. \_cursor = None
12. def get\_connection(self):
14. if not self.\_connection:
15. self.\_connection = psycopg.connect("user=ivan password=12345678 host=ubuntu.ivan.zeleniak.net port=5432 dbname=Parking",row\_factory=dict\_row)
16. return self.\_connection

19. def get\_cursor(self):
21. if self.\_cursor:
22. return \_cursor
23. else:
24. connection = self.get\_connection()
25. self.\_cursor = connection.cursor(row\_factory=dict\_row)
26. return self.\_cursor


30. def get\_make\_by\_name (self ,makeName):
31. cursor = self.get\_cursor()
32. query = 'SELECT \* FROM public."Makes" WHERE "MakeName" ='+"'" + str(makeName)+"'"
33. cursor.execute(query)
34. \_connection.commit()
35. record = cursor.fetchone()
36. return record

39. def get\_make\_by\_id (self ,makeID):
40. cursor = self.get\_cursor()
41. query = 'select \* from public."Makes" where "MakeID" ='+"'" + str(makeID)+"'"
42. cursor.execute(query)
43. \_connection.commit()
44. record = cursor.fetchone()
45. return record


49. def create\_make (self ,makeName,MakeDescription=''):
50. cursor = self.get\_cursor()
51. record = get\_make\_by\_name(makeName)
52. if cursor.rowcount == 0:
53. query = 'INSERT INTO public."Makes"("MakeName", "MakeDescription")' + "VALUES ('"+ str(makeName)+"','"+ str(MakeDescription)+"');"
54. cursor.execute(query)
56. \_connection.commit()
57. else:
58. makeid = record["MakeID"]
59. return makeid
61. def read\_make (self ,):
62. cursor = self.get\_cursor()
63. query = 'SELECT "MakeID", "MakeName", "MakeDescription" FROM public."Makes";'
64. cursor.execute(query)
65. \_connection.commit()
66. record = cursor.fetchall()
67. return record
69. def update\_make (self ,MakeID,MakeName,MakeDescription=''):
70. cursor = self.get\_cursor()
71. record = get\_make\_by\_id(MakeID)
72. if cursor.rowcount > 0:
73. query = 'UPDATE public."Makes" SET "MakeName"='+"'"+str(MakeName) +"'"+', "MakeDescription"='+"'"+str(MakeDescription) +"'"+ ' WHERE "MakeID" = '+ str(MakeID)+ ' ;'
74. cursor.execute(query)
75. \_connection.commit()

78. def del\_make (self ,MakeID):
79. cursor = self.get\_cursor()
80. record = get\_make\_by\_id(MakeID)
81. if cursor.rowcount > 0:
82. query = 'DELETE FROM public."Makes" WHERE "MakeID"='+ str(MakeID)+ ';'
83. cursor.execute(query)
84. \_connection.commit()


88. *##############Type*

91. def get\_CarType\_by\_name (self ,CarTypeName):
92. cursor = self.get\_cursor()
93. query = 'SELECT \* FROM public."CarType" WHERE "CarTypeName" ='+"'" + str(CarTypeName)+"'"
94. cursor.execute(query)
95. \_connection.commit()
96. record = cursor.fetchone()
97. return record

100. def get\_CarType\_by\_id (self ,CarTypeID):
101. cursor = self.get\_cursor()
102. query = 'select \* from public."CarType" where "CarTypeID" ='+"'" + str(CarTypeID)+"'"
103. cursor.execute(query)
104. \_connection.commit()
105. record = cursor.fetchone()
106. return record


110. def create\_CarType (self ,CarTypeName):
111. cursor = self.get\_cursor()
112. record = get\_CarType\_by\_name(CarTypeName)
113. if cursor.rowcount == 0:
114. query = 'INSERT INTO public."CarType"("CarTypeName")' + "VALUES ('"+ str(CarTypeName)+"');"
115. cursor.execute(query)
116. \_connection.commit()

119. def read\_CarType (self ,):
120. cursor = self.get\_cursor()
121. query = 'SELECT "CarTypeID", "CarTypeName" FROM public."CarType" ; '
122. cursor.execute(query)
123. \_connection.commit()
124. record = cursor.fetchall()
125. return record
127. def update\_CarType (self ,CarTypeID,CarTypeName):
128. cursor = self.get\_cursor()
129. record = get\_CarType\_by\_id(CarTypeID)
130. if cursor.rowcount > 0:
131. query = 'UPDATE public."CarType" SET "CarTypeName"='+"'"+str(CarTypeName) +"'"+' WHERE "CarTypeID" = '+ str(CarTypeID)+ ' ;'
132. cursor.execute(query)
133. \_connection.commit()

136. def del\_CarType (self ,CarTypeID):
137. cursor = self.get\_cursor()
138. record = get\_CarType\_by\_id(CarTypeID)
139. if cursor.rowcount > 0:
140. query = 'DELETE FROM public."CarType" WHERE "CarTypeID"='+ str(CarTypeID)+ ';'
141. cursor.execute(query)
142. \_connection.commit()


146. *########Cars*
148. def get\_Car\_by\_Reg\_num (self ,CarRegNum):
149. cursor = self.get\_cursor()
150. query = 'SELECT \* FROM public."Cars" WHERE "CarRegNum" ='+"'" + str(CarRegNum)+"'"
151. cursor.execute(query)
152. \_connection.commit()
153. record = cursor.fetchone()
154. return record



159. def create\_Car (self ,CarRegNum,CarMakeID,CarType):
160. cursor = self.get\_cursor()
161. record = get\_Car\_by\_Reg\_num(CarRegNum)
162. if cursor.rowcount == 0:
163. query = 'INSERT INTO public."Cars"("CarRegNum", "CarMakeID" , "CarType")' + "VALUES ('"+ str(CarRegNum)+"',"+ str(CarMakeID)+","+ str(CarType) +");"
164. cursor.execute(query)
166. \_connection.commit()
167. else:
168. makeid = record[0]
170. def read\_Car (self ,):
171. cursor = self.get\_cursor()
172. query = 'SELECT "CarRegNum", "CarMakeID", "CarType" FROM public."Cars";'
173. cursor.execute(query)
174. \_connection.commit()
175. record = cursor.fetchall()
176. return record
178. def update\_Car (self ,CarRegNum,CarMakeID,CarType):
179. cursor = self.get\_cursor()
180. record = get\_Car\_by\_Reg\_num(CarRegNum)
181. if cursor.rowcount > 0:
182. query = 'UPDATE public."Cars" SET "CarType"='+"'"+str(CarType) +"'"+', "CarMakeID"='+"'"+str(CarMakeID) +"'"+ ' WHERE "CarRegNum" = '+ str(CarRegNum)+ ' ;'
183. cursor.execute(query)
184. \_connection.commit()

187. def del\_Car (self ,CarRegNum):
188. cursor = self.get\_cursor()
189. record = get\_Car\_by\_Reg\_num(CarRegNum)
190. if cursor.rowcount > 0:
191. query = 'DELETE FROM public."Cars" WHERE "CarRegNum"='+ str(CarRegNum)+ ';'
192. cursor.execute(query)
193. \_connection.commit()

196. *##########Parking*

199. def get\_parking\_place\_by\_id (self,ParkingPlaceID):
200. cursor = self.get\_cursor()
201. query = 'select \* from public."ParkingPlace" where "ParkingPlaceID" ='+"'" + str(ParkingPlaceID)+"'"
202. cursor.execute(query)
203. \_connection.commit()
204. record = cursor.fetchone()
205. return record

208. def create\_parking\_place (self,ParkingPlaceID,ParkingPlaceDesc=''):
209. cursor = self.get\_cursor()
210. record = get\_parking\_place\_by\_id(ParkingPlaceID)
211. if cursor.rowcount == 0:
212. query = 'INSERT INTO public."ParkingPlace"("ParkingPlaceID", "ParkingPlaceDesc")' + "VALUES ('"+ str(ParkingPlaceID)+"','"+ str(ParkingPlaceDesc)+"');"
213. cursor.execute(query)
214. ParkingPlaceid=cursor.lastrowid
215. \_connection.commit()
216. else:
217. ParkingPlaceid = record[0]
218. return ParkingPlaceid
220. def read\_parking\_place (self,):
221. cursor = self.get\_cursor()
222. query = 'SELECT "ParkingPlaceID", "ParkingPlaceDesc" FROM public."ParkingPlace";'
223. cursor.execute(query)
224. \_connection.commit()
225. record = cursor.fetchall()
226. return record
228. def update\_parking\_place (self,ParkingPlaceID,ParkingPlaceDesc=''):
229. cursor = self.get\_cursor()
230. record = get\_parking\_place\_by\_id(ParkingPlaceID)
231. if cursor.rowcount > 0:
232. query = 'UPDATE public."ParkingPlace" SET ' +', "ParkingPlaceDesc"='+"'"+str(ParkingPlaceDesc) +"'"+ ' WHERE "ParkingPlaceID" = '+ str(ParkingPlaceID)+ ' ;'
233. cursor.execute(query)
234. \_connection.commit()

237. def del\_parking\_place (self,ParkingPlaceID):
238. cursor = self.get\_cursor()
239. record = get\_parking\_place\_by\_id(ParkingPlaceID)
240. if cursor.rowcount > 0:
241. query = 'DELETE FROM public."ParkingPlace" WHERE "ParkingPlaceID"='+ str(ParkingPlaceID)+ ';'
242. cursor.execute(query)
243. \_connection.commit()

246. *########Person*

249. def get\_person\_by\_last\_name (self,PersonLastName):
250. cursor = self.get\_cursor()
251. query = 'SELECT \* FROM public."Persons" WHERE "PersonLastName" ='+"'" + str(PersonLastName)+"'"
252. cursor.execute(query)
253. \_connection.commit()
254. record = cursor.fetchone()
255. return record


259. def get\_person\_by\_id (self,PersonID):
260. cursor = self.get\_cursor()
261. query = 'select \* from public."Persons" where "PersonID" ='+"'" + str(PersonID)+"'"
262. cursor.execute(query)
263. \_connection.commit()
264. record = cursor.fetchone()
265. return record
267. def get\_free\_parking\_place(self):
268. cursor = self.get\_cursor()
269. query = 'select "ParkingPlaceID", "ParkingPlaceDesc" from public."ParkingPlace" where "ParkingPlaceID" not in (select "ParkingID" from public."Contract" where "ContractEnd" >= (SELECT CURRENT\_DATE))'
270. cursor.execute(query)
271. \_connection.commit()
272. record = cursor.fetchall()
273. return record

276. def chek\_parking\_place(self,id):
277. cursor = self.get\_cursor()
278. query = 'select "ParkingPlaceID", "ParkingPlaceDesc" from public."ParkingPlace" where "ParkingPlaceID" not in (select "ParkingID" from public."Contract" where "ContractEnd" >= (SELECT CURRENT\_DATE)) and "ParkingPlaceID" = '+ str(id)
279. cursor.execute(query)
280. \_connection.commit()
281. record = cursor.fetchone()
282. return record
284. def create\_person (self,PersonID, PersonLastName, PersonName,PersonMidleName):
285. cursor = self.get\_cursor()
286. record = get\_person\_by\_id(PersonID)
287. if cursor.rowcount == 0:
288. query = 'INSERT INTO public."Persons"("PersonID", "PersonLastName", "PersonName" , "PersonMidleName" )' + "VALUES ('"+ str(PersonID), str(PersonLastName), str(PersonName),str(PersonMidleName) +"');"
289. cursor.execute(query)
291. \_connection.commit()
292. else:
293. makeid = record[0]

296. def read\_person (self,):
297. cursor = self.get\_cursor()
298. query = 'SELECT "PersonID", "PersonLastName", "PersonName","PersonMidleName" FROM public."Persons";'
299. cursor.execute(query)
300. \_connection.commit()
301. record = cursor.fetchall()
302. return record

305. def update\_person (self,PersonID, PersonLastName, PersonName,PersonMidleName):
306. cursor = self.get\_cursor()
307. record = get\_person\_by\_id(PersonID)
308. if cursor.rowcount > 0:
309. query = 'UPDATE public."Persons" SET "PersonLastName"='+"'"+str(PersonLastName) +"'"+'"PersonName"='+"'"+str(PersonName) +"'"+', "PersonMidleName"='+"'"+str(PersonMidleName) +"'"+ ' WHERE "PersonID" = '+ str(PersonID)+ ' ;'
310. cursor.execute(query)
311. \_connection.commit()


315. def del\_person (self,PersonID):
316. cursor = self.get\_cursor()
317. record = get\_person\_by\_id(PersonID)
318. if cursor.rowcount > 0:
319. query = 'DELETE FROM public."Persons" WHERE "PersonID"='+ str(PersonID)+ ';'
320. cursor.execute(query)
321. \_connection.commit()

324. *#######Phones*
325. def get\_phones\_by\_phones (self,Phone):
326. cursor = self.get\_cursor()
327. query = 'select \* from public."Phones" where "Phone" ='+"'" + str(Phone)+"'"
328. cursor.execute(query)
329. \_connection.commit()
330. record = cursor.fetchone()
331. return record



336. def create\_phones (self,PersonID,Phone):
337. cursor = self.get\_cursor()
338. record = get\_phones\_by\_phone(Phone)
339. if cursor.rowcount == 0:
340. query = 'INSERT INTO public."Phones"("PersonID", "Phone")' + "VALUES ('"+ str(PersonID)+"','"+ str(Phone)+"');"
341. cursor.execute(query)
343. \_connection.commit()
344. else:
345. makeid = record[0]
346. return makeid

349. def read\_phones (self,):
350. cursor = self.get\_cursor()
351. query = 'SELECT "PersonID", "Phone" FROM public."Phones";'
352. cursor.execute(query)
353. \_connection.commit()
354. record = cursor.fetchall()
355. return record

358. def update\_phones (self,PersonID,Phone):
359. cursor = self.get\_cursor()
360. record = get\_phones\_by\_phones(Phone)
361. if cursor.rowcount > 0:
362. query = 'UPDATE public."Phones" SET , "PersonID"='+"'"+str(PersonID) +"'"+ ' WHERE "Phone" = '+ str(Phone)+ ' ;'
363. cursor.execute(query)
364. \_connection.commit()


368. def del\_phones (self,Phone):
369. cursor = self.get\_cursor()
370. record = get\_phones\_by\_phones(Phone)
371. if cursor.rowcount > 0:
372. query = 'DELETE FROM public."Phones" WHERE "Phone"='+ str(Phone)+ ';'
373. cursor.execute(query)
374. \_connection.commit()

377. *####### Contracts*
379. def get\_contract\_by\_PersonID (self,PersonID):
380. cursor = self.get\_cursor()
381. query = 'SELECT \* FROM public."Contract" WHERE "PersonID" ='+"'" + str(PersonID)+"'"
382. cursor.execute(query)
383. \_connection.commit()
384. record = cursor.fetchone()
385. return record
387. def get\_currentContract\_by\_carRegNum(self,CarNumber):
388. cursor = self.get\_cursor()
389. query = '''select "ContractID", "CarNumber", "ParkingID", "Contract"."PersonID" as "PersonID", "ContractStart", "ContractEnd","CarMakeID","CarType", "CarTypeName", "MakeName", "PersonLastName", "PersonName", "PersonMidleName"
390. from public."Contract"
391. INNER JOIN public."Cars" on public."Contract"."CarNumber" = public."Cars"."CarRegNum"
392. INNER JOIN public."CarType" on public."Cars"."CarType" = public."CarType"."CarTypeID"
393. INNER JOIN public."Makes" on public."Makes"."MakeID" = public."Cars"."CarMakeID"
394. INNER JOIN public."Persons" on public."Persons"."PersonID" = public."Contract"."PersonID"
395. where "ContractID"=(select max("ContractID") from public."Contract" where "CarNumber" = \'''' + str(CarNumber)+"')"
396. cursor.execute(query)
397. \_connection.commit()
398. record = cursor.fetchone()
399. return record
401. def get\_contract\_by\_ParkingID (self,ParkingID):
402. cursor = self.get\_cursor()
403. query = 'select \* from public."Contract" where "ParkingID" ='+"'" + str(ParkingID)+"'"
404. cursor.execute(query)
405. \_connection.commit()
406. record = cursor.fetchone()
407. return record
409. def close\_contract(self,id):
410. cursor = self.get\_cursor()
411. query = 'UPDATE public."Contract" SET "ContractEnd"= (SELECT CURRENT\_DATE-1) WHERE "ContractID" = '+ str(ContractID)+ ' ;'
412. cursor.execute(query)
413. \_connection.commit()
415. def get\_contract\_by\_ContractID (self,ContractID):
416. cursor = self.get\_cursor()
417. query = 'select \* from public."Contract" where "ContractID" ='+"'" + str(ContractID)+"'"
418. cursor.execute(query)
419. \_connection.commit()
420. record = cursor.fetchone()
421. return record

424. def clear\_data(self):
425. cursor = self.get\_cursor()
426. query = ''' select '' as "ContractID", '' as "CarNumber", '' as "ParkingID", '' as "PersonID", '' as "ContractStart", '' as "ContractEnd", '' as "CarMakeID", '' as "CarType", '' as "CarTypeName", '' as "MakeName", '' as "PersonLastName", '' as "PersonName", '' as "PersonMidleName"'''
427. cursor.execute(query)
428. \_connection.commit()
429. record = cursor.fetchone()
430. return record

433. def create\_contract (self,CarNumber,PersonID,ParkingID,ContractEnd):
434. cursor = self.get\_cursor()
435. query = 'INSERT INTO public."Contract"("CarNumber","PersonID","ParkingID","ContractStart","ContractEnd")' + "VALUES ('"+ str(CarNumber)+"',"+str(PersonID)+","+str(ParkingID)+","+"(SELECT CURRENT\_DATE) " +","+"(SELECT CURRENT\_DATE +"+str(days)+"));"
436. cursor.execute(query)
438. \_connection.commit()

441. def read\_contract (self,):
442. cursor = self.get\_cursor()
443. query = 'SELECT "CarNumber","PersonID","ParkingID","ContractStart","ContractEnd" FROM public."Contract";'
444. cursor.execute(query)
445. \_connection.commit()
446. record = cursor.fetchall()
447. return record

450. def update\_contract (self,ContractID,CarNumber,PersonID,ParkingID,ContractStart,ContractEnd):
451. cursor = self.get\_cursor()
452. record = get\_contract\_by\_ContractID(ContractID)
453. if cursor.rowcount > 0:
454. query = 'UPDATE public."Contract" SET "CarNumber"='+"'"+str(CarNumber) +"'"+', "PersonID"='+"'"+str(PersonID) +"'"+', "ParkingID"='+"'"+str(ParkingID)+"'"+ ', "ContractStart"='+"'"+str(ContractStart) +"'"+ ', "ContractEnd"='+"'"+str(ContractEnd) +"'"+' WHERE "ContractID" = '+ str(ContractID)+ ' ;'
455. cursor.execute(query)
456. \_connection.commit()



461. def del\_contract\_by\_temp\_table\_CarNum(self,CarNum):
462. cursor = self.get\_cursor()
463. query ='''DROP TABLE IF EXISTS public."TMP\_Contract";
464. CREATE TABLE IF NOT EXISTS public."TMP\_Contract"("ContractID" smallint NOT NULL,"CarNumber" character varying(8) COLLATE pg\_catalog."default" NOT NULL,"PersonID" bigint NOT NULL,"ParkingID" smallint NOT NULL,"ContractStart" date NOT NULL,"ContractEnd" date NOT NULL);
465. insert into public."TMP\_Contract" ("ContractID","PersonID","CarNumber","ParkingID","ContractStart","ContractEnd") select "ContractID","PersonID","CarNumber","ParkingID","ContractStart","ContractEnd" from public."Contract" where "CarNumber" = /' '''+ str(CarNum)+ ''' /'' ;
466. delete from public."Contract" where "ContractID" in (select distinct "ContractID" from public."TMP\_Contract" );
467. delete from public."Cars" where "CarRegNum" = /' '''+ str(CarNum)+ ''' /';
468. delete from public."Persons" where "PersonID" in (select distinct "PersonID" from public."TMP\_Contract" ) and "PersonID" not in (select distinct "PersonID" from public."Contract" where "PersonID" in (select distinct "PersonID" from public."TMP\_Contract" ) );
469. DROP TABLE public."TMP\_Contract";'''
471. cursor.execute(query)
472. \_connection.commit()
474. def del\_contract\_by\_temp\_table\_personID(self,personID):
475. \_connection= self.get\_connection()
476. cursor = self.get\_cursor()
477. query ='''DROP TABLE IF EXISTS public."TMP\_Contract";
478. CREATE TABLE IF NOT EXISTS public."TMP\_Contract"("ContractID" bigint NOT NULL,"CarNumber" character varying(8) COLLATE pg\_catalog."default" NOT NULL,"PersonID" bigint NOT NULL,"ParkingID" smallint NOT NULL,"ContractStart" date NOT NULL,"ContractEnd" date NOT NULL);
479. insert into public."TMP\_Contract" ("ContractID","PersonID","CarNumber","ParkingID","ContractStart","ContractEnd") select "ContractID","PersonID","CarNumber","ParkingID","ContractStart","ContractEnd" from public."Contract" where "PersonID"= ''' +str(personID)+ ''';
480. delete from public."Contract" where "ContractID" in (select distinct "ContractID" from public."TMP\_Contract" );
481. delete from public."Cars" where "CarRegNum" in (select distinct "CarRegNum" from public."TMP\_Contract" ) and "CarRegNum" not in (select distinct "CarRegNum" from public."Contract" where "CarRegNum" in (select distinct "CarRegNum" from public."TMP\_Contract" ) );
482. delete from public."Persons" where "PersonID" = ''' +str(personID)+ ''';
483. DROP TABLE public."TMP\_Contract";'''
484. cursor.execute(query)
485. \_connection.commit()
487. def del\_contract\_by\_temp\_table\_ContractID(self,CarNum):
488. \_connection= self.get\_connection()
489. cursor = self.get\_cursor()
490. query ='''DROP TABLE IF EXISTS public."TMP\_Contract";
491. CREATE TABLE IF NOT EXISTS public."TMP\_Contract"
492. (
493. "ContractID" smallint NOT NULL,
494. "CarNumber" character varying(8) COLLATE pg\_catalog."default" NOT NULL,
495. "PersonID" bigint NOT NULL,
496. "ParkingID" smallint NOT NULL,
497. "ContractStart" date NOT NULL,
498. "ContractEnd" date NOT NULL
499. );
500. insert into public."TMP\_Contract" ("ContractID","PersonID","CarNumber","ParkingID","ContractStart","ContractEnd") select "ContractID","PersonID","CarNumber","ParkingID","ContractStart","ContractEnd" from public."Contract" where "ContractID"='''+str(ContractID) +''';
501. delete from public."Contract" where "ContractID" in (select distinct "ContractID" from public."TMP\_Contract" );
502. delete from public."Cars" where "CarRegNum" in (select distinct "CarRegNum" from public."TMP\_Contract" ) and "CarRegNum" not in (select distinct "CarRegNum" from public."Contract" where "CarRegNum" in (select distinct "CarRegNum" from public."TMP\_Contract" ) );
503. delete from public."Persons" where "PersonID" in (select distinct "PersonID" from public."TMP\_Contract" ) and "PersonID" not in (select distinct "PersonID" from public."Contract" where "PersonID" in (select distinct "PersonID" from public."TMP\_Contract" ) );
504. DROP TABLE public."TMP\_Contract";'''
506. cursor.execute(query)
507. \_connection.commit()
509. def random\_create (self, param):
510. cursor = self.get\_cursor()
511. query = "CALL public.generate\_data("+str(value)+");"
512. cursor.execute(query)
513. \_connection.commit()