НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»
ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ
Кафедра системного програмування та спеціалізованих комп'ютерних систем

Розрахунково-графічна робота

з дисципліни
«Бази даних і засоби управління»

Група: КВ-21

Виконав: Червоноокий Д.

Оцінка:

Київ – 2024

Проектування бази даних та ознайомлення з базовими операціями СУБД PostgreSQL

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).

Деталізоване завдання:

1. Забезпечити можливість уведення/редагування/вилучення даних у таблицях бази даних з можливістю контролю відповідності типів даних атрибутів таблиць (рядків, чисел, дати/часу). Для контролю пропонується два варіанти: контроль при введенні (валідація даних) та перехоплення помилок (try..except) від сервера PostgreSQL при виконанні відповідної команди SQL. Особливу увагу варто звернути на дані таблиць, що мають зв'язок 1:N. При цьому з боку батьківської таблиці необхідно контролювати **вилучення** рядків за умови наявності даних у підлеглій таблиці. З точки зору підлеглої таблиці варто контролювати наявність відповідного рядка у батьківській таблиці при виконанні **внесення** нових даних. Унеможливити виведення програмою системних помилок на екрані шляхом їх перехоплення і адекватної обробки. Внесення даних виконується користувачем у консольному вікні програми.
2. Забезпечити можливість автоматичної генерації великої кількості даних у таблицях за допомогою вбудованих у PostgreSQL функцій роботи з псевдовипадковими числами. Дані мають бути згенерованими **не мовою програмування, а відповідним SQL-запитом**!

Кількість даних для генерування має вводити користувач з клавіатури. Для тесту взяти 100 000 записів для однієї-двох таблиць.

Особливу увагу слід звернути на відповідність даних вимогам зовнішніх ключів з метою уникнення помилок порушення обмежень цілісності (foreign key).

3. Для реалізації пошуку необхідно підготувати 3 запити, що включають дані з декількох таблиць і фільтрують (WHERE) та групують (GROUP BY) рядки за 3-4 атрибутами цих таблиць. Забезпечити можливість уведення конкретних значень констант для фільтрації з клавіатури користувачем. Крім того, після виведення даних необхідно вивести час виконання запиту у мілісекундах. Перевірити швидкодію роботи запитів на попередньо згенерованих даних.
4. Програмний код організувати згідно шаблону Model-View-Controller(MVC). Приклад організації коду згідно шаблону доступний за посиланням та його опис за даним посиланням. При цьому модель, подання та контролер мають бути реалізовані у окремих файлах. Для доступу до бази даних використовувати **лише мову SQL** (без ORM).

Вимоги до пункту №1 деталізованого завдання:

- лістинги та скріншоти результатів виконання операції вилучення запису батьківської таблиці та виведення вмісту дочірньої таблиці після цього вилучення, а якщо воно неможливе, то результат перехоплення помилки з виведенням повідомлення про неможливість такого видалення за наявності залежних даних. Причини помилок мають бути пояснені;
- лістинги та скріншоти результатів виконання операції вставки запису в дочірню таблицю та виведення повідомлення про її неможливість, якщо в батьківські таблиці немає відповідного запису.

Вимоги до пункту №2 деталізованого завдання:

- копії екрану (ілюстрації) з фрагментами згенерованих даних таблиць;
- копії SQL-запитів, що ілюструють генерацію при визначених вхідних параметрах.

Вимоги до пункту №3 деталізованого завдання:

- ілюстрації уведення пошукового запиту та результатів виконання запитів;
- копії SQL-запитів, що ілюструють пошук із зазначеними початковими параметрами.

Вимоги до пункту №4 деталізованого завдання:

- ілюстрації програмного коду модуля “Model”, згідно із шаблоном MVC. Надати короткий опис функцій модуля.

Модель «сутність-зв'язок» предметної галузі "Електронна база даних для зберігання інформації про автомобілі"

1. **Автомобіль (Car)**
   - **Призначення**: Представляє кожен окремий автомобіль у базі даних.
   - **Атрибути**:
     - CarID (Первинний ключ): Унікальний ідентифікатор для кожного автомобіля.
     - Make: Бренд або виробник автомобіля (наприклад, Toyota, Ford).
     - Model: Модель автомобіля (наприклад, Corolla, Mustang).
     - Year: Рік виробництва автомобіля.
     - VIN: Ідентифікаційний номер транспортного засобу, унікальний ідентифікатор, призначений кожному автомобілю.

2. **Власник (Owner)**
   - **Призначення**: Представляє осіб, які володіють автомобілями.
   - **Атрибути**:
     - OwnerID (Первинний ключ): Унікальний ідентифікатор для кожного власника.
     - FirstName: Ім'я власника.
     - LastName: Прізвище власника.
     - Phone: Контактний телефон власника.
     - Email: Електронна адреса власника.

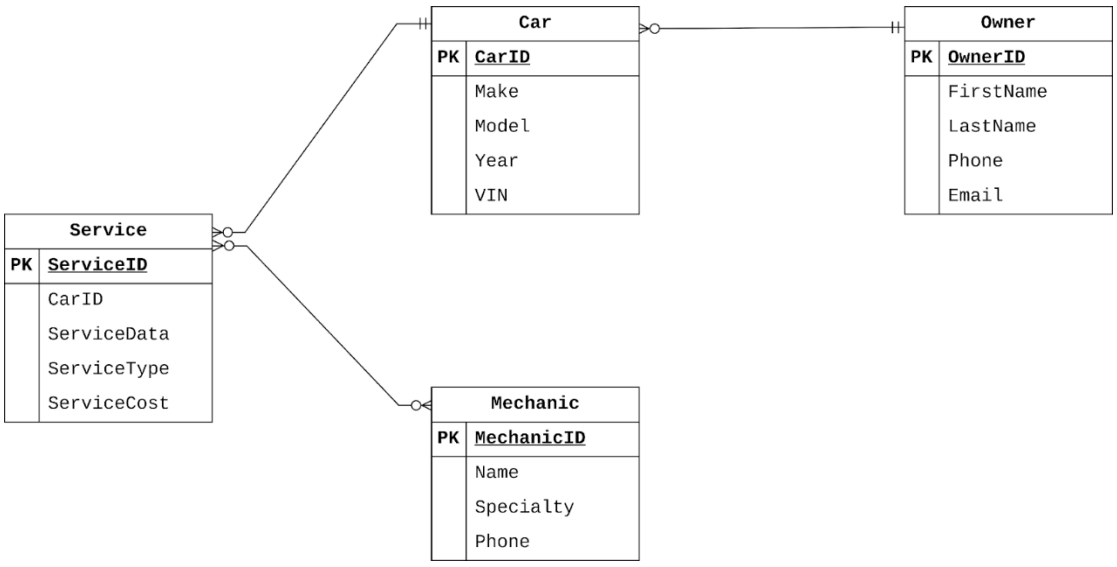
3. **Запис про обслуговування (Service)**
   - **Призначення**: Зберігає інформацію, пов'язану із записами про обслуговування автомобілів.
   - **Атрибути**:
     - ServiceID (Первинний ключ): Унікальний ідентифікатор для кожного запису про обслуговування.
     - CarID (Зовнішній ключ): Пов'язаний із сутністю Автомобіль.
     - ServiceDate: Дата, коли було виконано обслуговування.
     - ServiceType: Тип обслуговування (наприклад, заміна масла, ротація шин).

- ServiceCost: Вартість обслуговування.

4. **Механік (Mechanic)**
   - **Призначення**: Представляє механіків, які займаються обслуговуванням автомобілів.
   - **Атрибути**:
     - MechanicID (Первинний ключ): Унікальний ідентифікатор для кожного механіка.
     - Name: Ім'я механіка.
     - Specialty: Спеціалізація або галузь експертизи механіка (наприклад, ремонт двигуна, гальма).
     - Phone: Контактний телефон механіка.

5. **Взаємозв'язки (Relationships)**
   - **Автомобіль до Власника (1:N)**:
     - **Тип відносин**: **один-до-багатьох (1:N)**.
     - **Опис**: Автомобіль може мати лише одного поточного власника, але власник може володіти кількома автомобілями. Отже, зв'язок - Власник 1:N Автомобіль.
   - **Автомобіль до Запису про обслуговування (1:N)**:
     - **Тип відносин**: **один-до-багатьох (1:N)**.
     - **Опис**: Кожен автомобіль може мати декілька записів про обслуговування, але кожен запис про обслуговування стосується лише одного автомобіля. Отже, зв'язок - Автомобіль 1 :N Запис про обслуговування.
   - **Запис про обслуговування до Механіка (N:M)**:
     - **Тип відносин**: **багато-до-багатьох (N:M)**.
     - **Опис**: Запис про обслуговування може включати кілька механіків, і один механік може працювати над кількома записами про обслуговування. Оскільки N:M зв'язки не підтримуються безпосередньо в реляційних базах даних, я використаю перехідну таблицю (ServiceMechanic) для представлення цього зв'язку.
     - **Атрибути**:
       - ServiceID (Зовнішній ключ): Пов'язаний із сутністю Запис про обслуговування.
       - MechanicID (Зовнішній ключ): Пов'язаний із сутністю Механік.

- HoursWorked: Кількість годин, які механік працював над конкретним записом про обслуговування.



1. CRUD засобами консольного інтерфейсу

```
Menu:
1. Add Data
2. View Data
3. Update Data
4. Delete Data
5. Generate Random Data
6. Search Data
7. Quit
Enter your choice: 4

Select Table:
1. Car
2. Owner
3. Mechanic
4. ServiceRecord
5. ServiceMechanic
6. Back to Main Menu
Enter your choice: 2
Enter Owner ID: 1
Cannot delete Owner with associated Cars.
```

Власник не був видалений, так як до нього «прив’язані» автомобілі

```
Menu:
1. Add Data
2. View Data
3. Update Data
4. Delete Data
5. Generate Random Data
6. Search Data
7. Quit
Enter your choice: 1

Select Table:
1. Car
2. Owner
3. Mechanic
4. ServiceRecord
5. ServiceMechanic
6. Back to Main Menu
Enter your choice: 1
Enter car make: Toyota
Enter car model: Camry
Enter car year: 2019
Enter car VIN: 123456
Enter owner ID: 99
Owner with ID 99 does not exist.
```

Машина не була додана, так як власника з ID 99 не існує

2. Автоматичне пакетне генерування «рандомізованих» даних

```
Select Table:
1. Car
2. Owner
3. Mechanic
4. ServiceRecord
5. ServiceMechanic
6. Back to Main Menu
Enter your choice: 1
Enter the number of random records to generate:
100000
Generated 100000 random records for Car in
288363.85 ms.
```

	carid [PK] integer	make character varying (50)	model character varying (50)	year integer	vin character varying (17)	ownerid integer
89	91	d36f2d055b	35ef672806	262	e3d5b3ea88	10
90	92	466bb60841	f97dc52846	910	0aa0f78c34	11
91	93	2f2e55c665	6d329715e0	641	fe863a31f0	12
92	94	889e185742	4b55ad2c12	923	2c470e4aa5	6
93	95	48cf40f3a0	dd746d5c11	254	5f0caf1b8e	8
94	96	fc3d9619c8	2dd6cdd11f	283	8fb4a5fe60	5
95	97	b5423333b4	214ea862bf	517	0e1cfa9933	3
96	98	ff9098c7eb	95a91f8c88	485	3ea95b2511	3
97	99	6149de373d	af301abd56	812	6625528543	2
98	100	ab746c2330	d367cbf28a	509	8577f9de14	2
99	101	761fec17b8	a13267dfb9	464	aa1da56cef	2
100	102	ebbda9d94a	46cd61a6a6	738	c3428c415a	4

```
def generate_data(self, num_rows):
    """
    Generate random data for the table using SQL functions,
    handling various data types and foreign keys.
    """
    try:
      cursor = self.conn.cursor()

      # Get the list of columns excluding the primary key
      columns = self.columns.copy()
```

```
if self.pk in columns:
  columns.remove(self.pk)

# Retrieve data types for each column
data_types = []
for column in columns:
  cursor.execute("""
    SELECT data_type
    FROM information_schema.columns
    WHERE table_name=%s AND column_name=%s
  """, [self.table_name, column])
  data_type = cursor.fetchone()[0]
  data_types.append(data_type)

# Identify foreign key relationships
foreign_keys = {}
for column in columns:
  cursor.execute("""
    SELECT
      tc.constraint_name,
      kcu.column_name AS fk_column,
      ccu.table_name AS foreign_table,
      ccu.column_name AS foreign_column
    FROM
      information_schema.table_constraints AS tc
      JOIN information_schema.key_column_usage AS kcu
        ON tc.constraint_name = kcu.constraint_name
      JOIN information_schema.constraint_column_usage AS ccu
        ON ccu.constraint_name = tc.constraint_name
    WHERE
```

```
            tc.constraint_type = 'FOREIGN KEY'
            AND tc.table_name = %s
            AND kcu.column_name = %s
        """, [self.table_name, column])
        fk_info = cursor.fetchone()
        if fk_info:
          foreign_keys[column] = {
            'foreign_table': fk_info[2],
            'foreign_column': fk_info[3]
          }

    # Generate and execute INSERT statements for the specified
number of rows
    for _ in range(num_rows):
      # Build value expressions for each column
      value_expressions = []
      for colu
```

3. Реалізація пошуку за декількома атрибутами

```
Menu:
1. Add Data
2. View Data
3. Update Data
4. Delete Data
5. Generate Random Data
6. Search Data
7. Quit
Enter your choice: 6

Select Search Query:
1. Search Cars by Make and Year Range
2. Search Mechanics by Specialty and Name Pattern
3. Search Service Records by Date Range and
Service Type
4. Back to Main Menu
Enter your choice: 1
Enter car make (leave blank for any):
Enter start year (leave blank for any): 500
Enter end year (leave blank for any): 501
{'carid': 439, 'make': 'ccc848e211', 'model':
'e5c3f1a857', 'year': 501, 'vin': '4353f2ee44',
'ownerid': 4}
{'carid': 1589, 'make': 'a0bea04e53', 'model':
'7facdb7e08', 'year': 500, 'vin': '44ad42c9fc',
'ownerid': 5}
{'carid': 1728, 'make': '968ff6c961', 'model':
'2ae717c8fd', 'year': 501, 'vin': '0264a46432',
'ownerid': 7}
{'carid': 2223, 'make': 'bea67a698d', 'model':
'c9eded85b5', 'year': 501, 'vin': '6a532d7002',
'ownerid': 6}
{'carid': 2456, 'make': '968aae6c60', 'model':
'0aaa4c306d', 'year': 500, 'vin': '590d6b6b5f',
'ownerid': 4}
{'carid': 2529, 'make': '4d8c22022b', 'model':
'9ba188864b', 'year': 500, 'vin': 'b3b0d449b1',
'ownerid': 6}
{'carid': 2662, 'make': '56754cfcca', 'model':
'8e224cfdb0', 'year': 500, 'vin': '8490d9022e',
'ownerid': 7}
```

```
---

Select Search Query:
1. Search Cars by Make and Year Range
2. Search Mechanics by Specialty and Name Pattern
3. Search Service Records by Date Range and
Service Type
4. Back to Main Menu
Enter your choice: 2
Enter mechanic specialty (leave blank for any):
Двигун
Enter mechanic name pattern (e.g., '%John%'):
{'mechanicid': 1, 'name': 'Олександр Іванов',
'specialty': 'Двигун', 'phone': '0932233444'}
Query executed in 1.92 ms.

---

Select Search Query:
1. Search Cars by Make and Year Range
2. Search Mechanics by Specialty and Name Pattern
3. Search Service Records by Date Range and
Service Type
4. Back to Main Menu
Enter your choice: 3
Enter start date (YYYY-MM-DD, leave blank for
any): 2020-01-01
Enter end date (YYYY-MM-DD, leave blank for any):
2030-01-01
Enter service type (leave blank for any):
{'serviceid': 1, 'carid': 1, 'servicedate':
datetime.date(2023, 1, 15), 'servicetype':
'Заміна масла', 'servicecost': Decimal('800.00')}
{'serviceid': 2, 'carid': 2, 'servicedate':
datetime.date(2023, 2, 10), 'servicetype':
'Ремонт гальмівної системи', 'servicecost':
Decimal('1500.00')}
Query executed in 3.57 ms.

---
```

```
def search_data(self):

    while True:

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

```
        if choice == '1':
            criteria = self.view.get_car_search_input()
            car_model = Car()
            query = "SELECT * FROM car WHERE 1=1"
            params = []
            if criteria['make']:
                query += " AND make = %s"
                params.append(criteria['make'])
            if criteria['year_from']:
                query += " AND year >= %s"
                params.append(int(criteria['year_from']))
            if criteria['year_to']:
                query += " AND year <= %s"
                params.append(int(criteria['year_to']))
            start_time = time.time()
            try:
                with
car_model.conn.cursor(cursor_factory=psycopg2.extras.DictCursor)
as cursor:
                    cursor.execute(query, params)
                    records = cursor.fetchall()
                    end_time = time.time()
                    self.view.show_records(records)
                    self.view.show_message(f"Query executed in {(end_time -
start_time)*1000:.2f} ms.")
            except Exception as e:
                self.view.show_message(f"Error executing search: {e}")

        elif choice == '2':
            criteria = self.view.get_mechanic_search_input()
            mechanic_model = Mechanic()
```

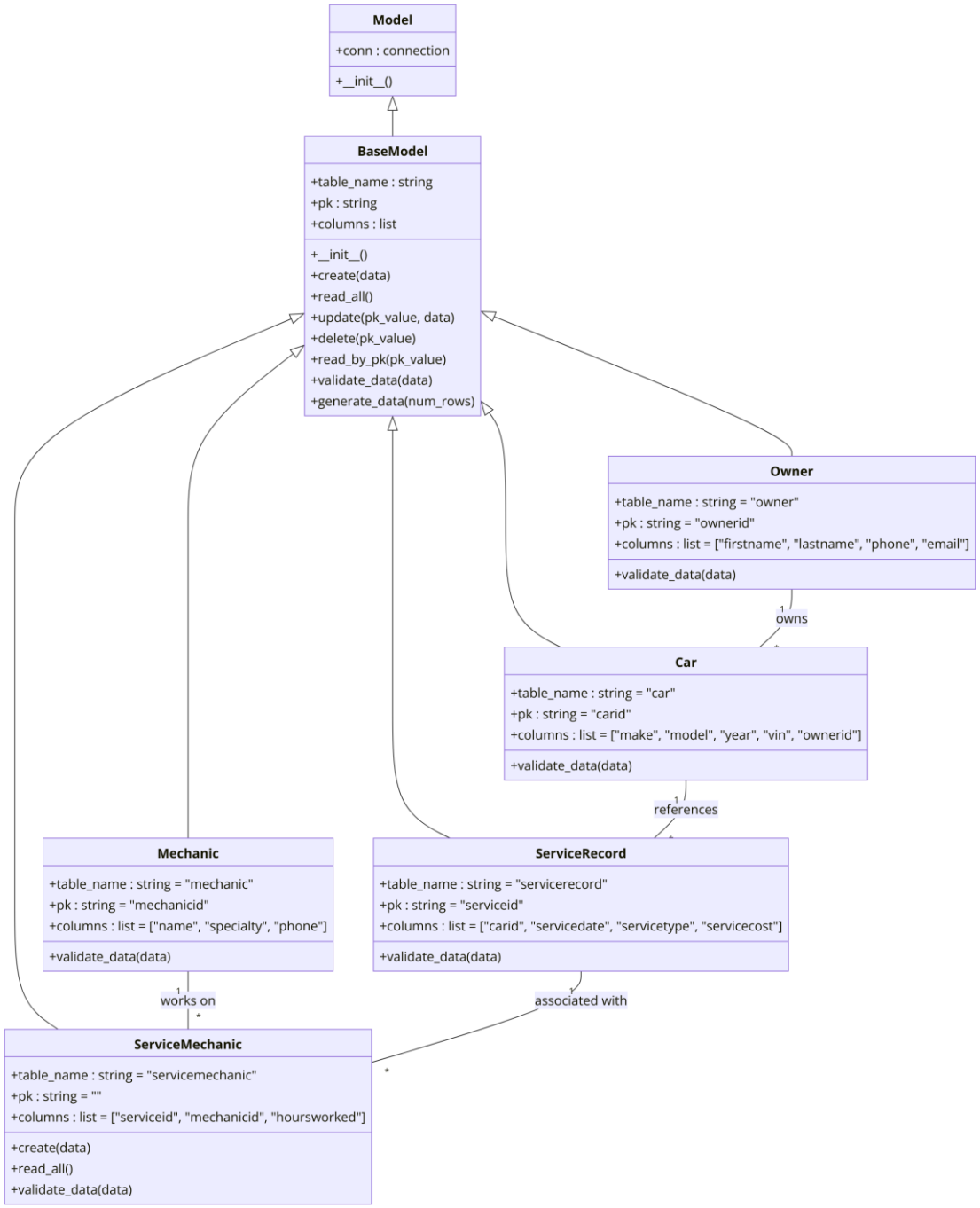
```
        query = "SELECT * FROM mechanic WHERE 1=1"
        params = []
        if criteria['specialty']:
          query += " AND specialty = %s"
          params.append(criteria['specialty'])
        if criteria['name_pattern']:
          query += " AND name LIKE %s"
          params.append(criteria['name_pattern'])
        start_time = time.time()
        try:
          with
mechanic_model.conn.cursor(cursor_factory=psycopg2.extras.DictCur
sor) as cursor:
            cursor.execute(query, params)
            records = cursor.fetchall()
            end_time = time.time()
            self.view.show_records(records)
            self.view.show_message(f"Query executed in {(end_time -
start_time)*1000:.2f} ms.")
        except Exception as e:
          self.view.show_message(f"Error executing search: {e}")

      elif choice == '3':
        criteria = self.view.get_service_record_search_input()
        service_model = ServiceRecord()
        query = "SELECT * FROM servicerecord WHERE 1=1"
        params = []
        if criteria['date_from']:
          query += " AND servicedate >= %s"
          params.append(criteria['date_from'])
        if criteria['date_to']:
```

```
            query += " AND servicedate <= %s"
            params.append(criteria['date_to'])
        if criteria['servicetype']:
            query += " AND servicetype = %s"
            params.append(criteria['servicetype'])
        start_time = time.time()
        try:
            with
service_model.conn.cursor(cursor_factory=psycopg2.extras.DictCurs
or) as cursor:
                cursor.execute(query, params)
                records = cursor.fetchall()
                end_time = time.time()
                self.view.show_records(records)
                self.view.show_message(f"Query executed in {(end_time -
start_time)*1000:.2f} ms.")
        except Exception as e:
            self.view.show_message(f"Error executing search: {e}")

    elif choice == '4':
        break
    else:
        self.view.show_message("Invalid choice.")
```

4. MVC


Model
+conn : connection
+__init__()
BaseModel
+table_name : string
+pk : string
+columns : list
+__init__()
+create(data)
+read_all()
+update(pk_value, data)
+delete(pk_value)
+read_by_pk(pk_value)
+validate_data(data)
+generate_data(num_rows)
Owner
+table_name : string = "owner"
+pk : string = "ownerid"
+columns : list = ["firstname", "lastname", "phone", "email"]
+validate_data(data)
1
owns
Car
+table_name : string = "car"
+pk : string = "carid"
+columns : list = ["make", "model", "year", "vin", "ownerid"]
+validate_data(data)
1
references
Mechanic
+table_name : string = "mechanic"
+pk : string = "mechanicid"
+columns : list = ["name", "specialty", "phone"]
+validate_data(data)
ServiceRecord
+table_name : string = "servicerecord"
+pk : string = "serviceid"
+columns : list = ["carid", "servicedate", "servicetype", "servicecost"]
+validate_data(data)
1
works on
*
1
associated with
*
ServiceMechanic
+table_name : string = "servicemechanic"
+pk : string = ""
+columns : list = ["serviceid", "mechanicid", "hoursworked"]
+create(data)
+read_all()
+validate_data(data)

Model: Базовий клас для встановлення з'єднання з базою даних.

BaseModel: Узагальнений клас для всіх моделей бази даних, що містить основні CRUD операції:

- `create(data)`: Додає новий запис у таблицю.
- `read_all()`: Отримує всі записи таблиці (обмежено 100 записами).
- `update(pk_value, data)`: Оновлює запис за його первинним ключем.
- `delete(pk_value)`: Видаляє запис за первинним ключем.
- `read_by_pk(pk_value)`: Отримує запис за первинним ключем.
- `generate_data(num_rows)`: Генерує випадкові дані для таблиці, використовуючи SQL функції, з урахуванням типів даних та зовнішніх ключів.

Car, Owner, Mechanic, ServiceRecord, ServiceMechanic: Спеціалізовані моделі для таблиць `car`, `owner`, `mechanic`, `servicerecord` та `servicemechanic`. Вони успадковують `BaseModel` та додають:

- `validate_data(data)`: Перевіряє відповідність вхідних даних, наприклад, існування зовнішніх ключів (`carid`, `ownerid`, `mechanicid` та інші).

Розроблено на базі:
python3
psycopg2

github: https://github.com/ChervonookyiDmytro/DatabasesRGR/tree/main
telegram: https://t.me/MyGTOFantasy