**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

**“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**

**ІМЕНІ ІГОРЯ СІКОРСЬКОГО”**

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

**КУРСОВА РОБОТА**

з дисципліни “Бази даних”

спеціальність 121 – Програмна інженерія

на тему: **Система аналізу рентабельності продуктів інтернет магазинів**

(назва теми)

|  |  |  |
| --- | --- | --- |
| **Студент групи** КП-92 | **Фенченко Ігор Юрійович**  (ПІБ) | \_\_\_\_\_\_\_\_\_  (підпис) |
| **Викладач**  **к.т.н, доцент кафедри СПіСКС** | **Петрашенко А.В.** | \_\_\_\_\_\_\_\_\_  (підпис) |

Захищено з оцінкою \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Київ – 2020

### **Анотація**

Метою розробки даного курсового проекту є набуття практичних навичок розробки сучасного програмного забезпечення, що взаємодіє з

реляційними базами даних та навичок оформлення відповідного текстового, програмного та ілюстративного матеріалу у формі проектної документації.

У результаті виконання курсового проекту мною було опановано розроблення програмного забезпечення для реляційних баз даних, володіння основами використання СУБД, засобами аналізу великих обсягів даних, генерації та отримання даних з мережі Інтернет, індексації таблиць бази даних та роботи з ними.

Темою даного курсового проекту є створення системи аналізу рентабельності продуктів інтернет-магазину на прикладі даних з інтернет магазину Amazon, так як багато даних цього ресурсу розміщено у вільному доступі, що допомагає зробити дані для аналізу більш близькими до реальності, та роботи з підтримання цієї системи.

Галуззю застосування даної розробки є середній та малий бізнес .

У документі викладена проблематика та актуальність, аналіз використаного інструментарію, описана структура бази даних, опис розробленого програмного забезпечення, способів прискорення роботи з СУБД, забезпечення реплікації та резервного копіювання даних в БД.

Результатами даного проекту став програмний додаток для роботи з, наближеною до реальної, базою даних інтернет-магазину, оновлення та підтримання даної БД актуальною інформацією та прискорення швидкодії отримання даних з неї.

### **Зміст**

[**Анотація**](#_rzlbtqj5l5e9) **2**

[**Зміст**](#_3fikipmiqfpx) **3**

[**Вступ**](#_1oq1r8x21xm7) **4**

[**Аналіз інструментарію для виконання курсової роботи**](#_4o93jc82tbqy) **5**

[Обґрунтування вибору мови програмування:](#_tkr24dkwurg0) 5

[Обґрунтування вибору бібліотек та фреймворків](#_9qcrifdvn0ad) 5

[Обґрунтування вибору СУБД](#_t652yf797gsj) 6

[Обґрунтування вибору середовища розробки](#_k8ovhq7n6jvy) 7

[**Структура бази даних**](#_m5vf3xj37sz) **8**

[**Опис програмного забезпечення**](#_t6zxh3ius5ge) **10**

[Загальна структура програмного забезпечення](#_ulfmobnbehum) 10

[Опис модулів програмного забезпечення](#_y64b1y3pxbh) 10

[Опис основних алгоритмів роботи](#_5s1brcx42zgo) 13

[**Аналіз функціонування засобів реплікації**](#_jco5cnvl9dq1) **14**

[**Аналіз функціонування засобів резервування/відновлення**](#_k5uu3rs1t6ah) **15**

[**Аналіз результатів підвищення швидкодії виконання запитів**](#_at9owiygb3kq) **17**

[**Опис результатів аналізу предметної галузі**](#_3a2yjv44h9og) **18**

[**Висновки**](#_rgkyk72kxmdb) **20**

[**Література**](#_lbt38tcckn56) **21**

[**Додатки**](#_uu7mv64wev8f) **23**

[Графічні матеріали](#_z1pw2koyv6m6) 23

[Фрагменти програмного коду](#_8kclru35nv3u) 31

### **Вступ**

В сучасному світі, підприємцям, що мають свої магазини, життєво необхідно мати інформацію про споживання та тенденції, щодо товарів, які є в їх магазині. Очевидно, що зараз кожен такий магазин має свою електронну базу даних. Проте, часто, дані про покупки та замовлення жодним чином не аналізуються, що призводить до втрати прибутків та неефективного ведення бізнесу.

Для вирішення цієї проблеми, було розроблено додаток, який має на меті вирішення проблем аналізу, керування та фільтрації даних, що містяться в БД інтернет-магазинів.

Галуззю застосування даної розробки є середній та малий бізнес.

Метою розробки даного курсового проекту є набуття навичок розробки сучасного програмного забезпечення, що взаємодіє з реляційними базами даних, володіння основами використання СУБД, засобами аналізу великих обсягів даних, генерації та отримання даних з мережі Інтернет, індексації таблиць бази даних, роботи з ними, зберігання та реплікації даних засобами СУБД.

У результаті виконання курсового проекту отримано програмне забезпечення для реляційних баз даних.

Дані для аналізу були взяті з датасетів та скрапінгу онлайн сервісу Amazon, що забезпечує актуальність та наближеність даних до реальної ситуації. Так як Amazon не надає інформації про замовлення та користувачів інтернет-магазину, було реалізовано псевдо-випадкову генерацію персональних даних користувачів, засобами СУБД.

### Аналіз інструментарію для виконання курсової роботи

#### Обґрунтування вибору мови програмування:

Мовою програмування для даного проекту було обрано Python 3 через велику кількість відкритих бібліотек та фреймворків, що спрощує отримання бажаного результату, зокрема NumPy (бібліотека для роботи із великими масивами даних), Matplotlib ( бібліотека для візуалізації даних у вигляді 2D і 3D графіків), Pandas (бібліотека, для структуризації роботи із масивами даних), sqlalchemy (ORM для роботи з СУБД PostgreSQL) та інших ресурсів.

Також, важливою перевагою Python, є її динамічна типізація та можливість отримати всю інформацію про об’єкт в програмному коді за допомогою властивості \_\_dict\_\_, що уможливлює створення ПЗ для автоматизації деяких процесів та скорочення кількості модулів.

Python дозволяє швидко впоратися з задачею обробки даних у форматі CSV, що не завжди оформлені корректно та однотипно та надає можливості для створення скраперів, щоб отримувати інформацію з інтернет-ресурсів.

Окрім цього, Python є абсолютним лідером по використанню у Data Science. [1]

#### Обґрунтування вибору бібліотек та фреймворків

Автором проекту було використано наступні бібліотеки та фреймворки:

* NumPy — надає підтримку великих багатовимірних масивів і матриць та операцій над ними. Використовується для обробки даних для візуалізації даних. [2]
* Matplotlib — бібліотека Python 2D, яка представляє числові дані у різноманітних форматах та інтерактивних середовищах на різних платформах. Також ця бібліотека - математичне розширення NumPy. Засоби даної бібліотеки дозволили легко побудувати потрібні діаграми та графіки для візуального представлення. [3]
* Pandas — бібліотека для обробки та аналізу даних. Працює разом з NumPy та Matplotlib для легкої та швидкої обробки даних для візуалізації даних. [4]
* Psycopg2 — бібліотека для роботи з СУБД PostgreSQL. Потрібна для роботи ORM sqlalchemy. [5]
* Sqlalchemy — ORM для роботи з СУБД PostgreSQL. Потрібна для автоматизації запитів, створення моделей, покращення швидкодії виконання запитів та покращення вигляду коду. [6]
* СUI constructor — бібліотека, створена автором даного проекту, для зручного та швидкого створення консольного інтерфейсу. [7]
* Selectorlib — бібліотека для створення екстракторів, з метою створення ПЗ для скрапінгу веб-сторінок. [8]

#### *Обґрунтування вибору СУБД*

Було обрано PostgreSQL, як найбільш відповідну СУБД для даного проекту. Ця СУБД є масштабованою і здатна обробляти терабайти даних, має підтримку формату JSON, має безліч визначених функцій і можливостей індексації та є реляційною, що дуже важливо для складної системи зв’язків сутностей таблиць БД інтернет-магазину.

Також, слід зазначити, що PostgreSQL є безкоштовною СУБД, з відкритим кодом, а сервер PostgreSQL працює на ОС Лінукс, на якій проводилася розробка проекту. Дана СУБД надає можливості легко створювати і керувати індексами, тригерами, поточні реплікації та резервування БД. [9]

#### *Обґрунтування вибору середовища розробки*

Розробка проводилась на операційній системі Лінукс, проте проект підтримує і інші платформи, зокрема Windows.

В якості IDE було обрано IDE PyCharm, яке надає можливості підключення та роботи з СУБД PostgreSQL. [10]

### Структура бази даних

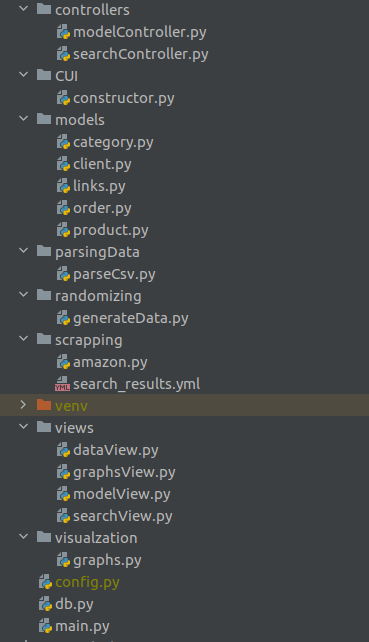
База даних проекту включає в себе 6 таблиць:

1. Product - основна таблиця проекту. Містить інформацію про певний продукт інтернет-магазину. Ця таблиця підлягає основному аналізу в проекті, тому її заповнення забезпечують парсинг і скрапінг інформації з ресурсу Amazon. Поля таблиці:
   1. id (int) - первинний ключ
   2. name (varchar) - назва продукту
   3. brand (varchar) - назва бренду, який характеризує продукт
   4. cost (numeric) - ціна продукту
   5. manufacturer (varchar) - компанія, яка виготовила цей продукт
   6. manufacture\_date (date) - дата виробництва
2. Order - таблиця, яка характеризує замовлення певного користувача магазину. Заповнюється за допомогою генерації псевдовипадкової інформації. Поля таблиці:
   1. id (int) - первинний ключ
   2. transaction\_date (date) - дата замовлення
   3. taxes\_sum (numeric) - сума податків, для цього замовлення
   4. client\_id (int) - FK, вторинний ключ, який вказує на поле id таблиці Client і характеризує клієнта, який зробив дане замовлення
3. Client - таблиця, яка характеризує клієнта інтернет-магазину. Заповнюється за допомогою генерації псевдовипадкової інформації. Поля таблиці:
   1. id (int) - первинний ключ
   2. name (varchar) - ім’я користувача
   3. birthday\_date (date) - дата народження користувача
   4. email (varchar) - електронна адреса користувача
4. Category - таблиця, що характеризує категорію, якій належить кожний продукт. Також підлягає основному аналізу, тому заповнюється даними з датасетів та ресурсів Amazon. Поля таблиці:
   1. id (int) - первинний ключ
   2. name (varchar) - назва категорії
   3. type (varchar) - тип категорії
5. Link\_Product-Category - таблиця, створена для приведення БД до 3 нормальної форми. Є допоміжною таблицею для реалізації Many to Many зв’язку між таблицями Product і Category. Поля таблиці:
   1. product\_id (int) - FK, вторинний ключ, який вказує на поле id таблиці Product
   2. category\_id (int) - FK, вторинний ключ, який вказує на поле id таблиці Category
6. Link\_Product-Order - таблиця, створена для приведення БД до 3 нормальної форми. Є допоміжною таблицею для реалізації Many to Many зв’язку між таблицями Product і Order. Поля таблиці:
   1. product\_id (int) - FK, вторинний ключ, який вказує на поле id таблиці Product
   2. order\_id (int) - FK, вторинний ключ, який вказує на поле id таблиці Order

Усі таблиці є нормалізованими до 3 нормальної форми. Див. додаток A.1

### Опис програмного забезпечення

#### Загальна структура програмного забезпечення



#### Опис модулів програмного забезпечення

* controllers - розділ, що містить контролери проекту. Контролери працюють безпосередньо з БД, за допомогою ORM. Мають обробку помилок, повертають дані представленню.
  + modelController - універсальний контролер, для будь-якої моделі ORM. Забезпечує CRUD операції над сутностями, пагінацію результатів та має методи для отримання полів сутності, над якою виконує операції.
  + searchController - контролер для забезпечення виконання специфічних запитів до БД з метою отримання корисної інформації.
* views - розділ, що містить представлення проекту. Представлення працюють з користувачем проекту засобами консольного інтерфейсу, повідомляють йому про помилки та швидкість виконання запитів. Забезпечують безпечний ввід користувачем даних, шляхом їх валідації.
  + modelView - універсальне представлення для будь-якої моделі ORM. Працює з екземплярами modelController.
  + searchView - представлення корисних пошукових запитів. Працює з searchController.
  + dataView - представлення для роботи з генерацією даних. Має опції до виконання функцій, що ініціалізують парсинг, скрапінг та генерацію псевдовипадкової інформації. Працює з модулями amazon, parseCsv, generateData.
  + graphsView - представлення для показу графіків та інших продуктів функцій для графічного аналізу сутностей БД. Працює з модулем graphs.
* models
  + category - модуль, що описує модель ORM Category, її створення, формат відповіді серверу Postgres та зв’язки цієї моделі з іншими моделями ORM.
  + client - модуль, що описує модель ORM Client, її створення, формат відповіді серверу Postgres та зв’язки цієї моделі з іншими моделями ORM.
  + order - модуль, що описує модель ORM Order, її створення, формат відповіді серверу Postgres та зв’язки цієї моделі з іншими моделями ORM.
  + product - модуль, що описує модель ORM Product, її створення, формат відповіді серверу Postgres та зв’язки цієї моделі з іншими моделями ORM.
  + links - модуль, що описує таблиці взаємозв’язків Many to Many, потрібні для роботи ORM.
* Інші модулі
  + amazon - модуль, для скрапінгу сервісу Amazon. Забезпечує збирання інформації про продукти за певною категорією.
  + parseCsv - модуль, відповідальний за парсинг датасетів.
  + generateData - модуль для генерації псевдовипадкової інформації.
  + Graphs - модуль, що забезпечує візуалізацію даних БД.
  + constructor - модуль для створення консольного інтерфейсу користвуча.
  + main - стартовий модуль прорами, містить опис головного меню.
  + db - модуль, що відповідає за підключення до серверу БД, створення екземпляру класу Base для подальшого опису моделей ORM та коду, що ініціалізує користувацькі функції, індекси та тригери. Містить функцію recreate\_database для очищення БД.
  + config - модуль, що містить персональні дані користувача системи для підключення до БД.

#### Опис основних алгоритмів роботи

* Пересування по меню здійснюється клавішами W, S або стрілками вверх / вниз. Виконання пункту меню викликається натисканням кнопки Enter на поточному пункті меню. Вивід помилок, або повідомлень користувачу виконується біля назви меню зверху.
* Парсинг CSV відбувається засобами стандартних бібліотек Python.
* Скрапінг відбувається за допомогою зчитування даних напряму з html / css розмітки сайту. Для цього поряд з модулем, що відповідає за скрапінг є yml файла з описом потрібних даних.
* Генерація псевдовипадкової інформації відбувається шляхом транзакції серверу БД з вказанням потрібної кількості рядків таблиці. Для цього проект при створенні ініціалізує користувацькі функції, потрібні для генерації.
* Модулі, що є універсальними, а саме modelView та modelController, працюють завдяки отримання інформації про екземпляр об’єкту з поля \_\_dict\_\_. Це поле є стандартним для об’єктів Python.
* Всі запити до БД здійснюються за допомогою ORM.
* Модуль для конструювання консольного інтерфейсу використовує як основу n-ary дерево. Його обхід алгоритмізовано.

### Аналіз функціонування засобів реплікації

Реплікація - це процес синхронізації даних на декількох серверах. Даний механізм зменшує витрати ресурсів і збільшує доступність даних,

копії яких зберігаються на різних серверах. Реплікація захищає базу даних від втрати єдиного сервера і дозволяє зберегти дані в разі технічної несправності на одному з серверів. У PostgeSQL реплікація досягається шляхом підняття резервного серверу на іншому порті, та налаштування потокової передачі на slave даних, що змінюються на основому сервері.

У разі технічних проблем на основному сервері, вторинний сервер, який являє собою копію, первинного забирає на себе обов’язки головного серверу і стає первинним сервером БД.

Щоб перевірити коректну роботу реплікацій, було аварійно завершено сервер, що розгортається на 5432 порті та є первинним. Вторинний сервер, що розгортається на порті 5433 став первинним і повністю забезпечив коректну роботу проекту.

Налаштування реплікації здійснювалось на ОС Linux за допомогою підняття додаткового серверу PostgreSQL на іншому порті, паралельно головному серверу та налаштування конфігураційних файлів postgresql.conf та pg\_hba.conf. Див. додатки А.2. [11]

Запуск сервера реплікації, за відсутності автоматичного включення при початку роботи основного серверу, відбувається на командою (Ubuntu 20.04, PostgreSQL 12):

|  |
| --- |
| sudo -u postgres /usr/lib/postgresql/12/bin/pg\_ctl -D /home/bujhmt/KPI/database/DB\_coursework/replica start |

### Аналіз функціонування засобів резервування/відновлення

Для забезпечення даного механізму, було обрано стандартну утиліту PostgeSQL pg\_dump та pg\_dumpall. Ідея цього методу в тому, що сервер БД генерує текстовий файл з командами SQL, які при виконанні на сервері створять БД в тому самому вигляді, якою вона була на момент створення резервної копії. [12]

Перевагою цього метода над створенням фізичних файлів CSV, або інших форматів, та архівуванням WAL є простота, гнучкість і ефективність, яка досягається тим, що:

1. При відновленні резервної копії не потрібно ніяк працювати з файлом: парсити його, або фільтрувати.
2. Копія не залежить від версії PostgreSQL, на відміну від архівування WAL.
3. Зробити копію можна в декілька кроків.

Виконання команд, що ініціалізують створення резервних копій (Ubuntu 20.04, PostgreSQL 12):

* Створення копії кластеру:

|  |
| --- |
| sudo /usr/lib/postgresql/12/bin/pg\_dumpall -U postgres > /home/bujhmt/KPI/database/DB\_coursework/backup/all.sql |

* Створення копії обраної БД:

|  |
| --- |
| sudo /usr/lib/postgresql/12/bin/pg\_dump -U postgres -W -F t coursework > /home/bujhmt/KPI/database/DB\_coursework/backup/main.tar |

Команди для відновлення БД створеними копіями:

* Відновлення кластеру:

|  |
| --- |
| psql -f /home/bujhmt/KPI/database/DB\_coursework/backup/all.sql postgres |

* Відновлення обраної БД:

|  |
| --- |
| psql coursework < /home/bujhmt/KPI/database/DB\_coursework/backup/main.tar |

Див. додатки A.3.

### Аналіз результатів підвищення швидкодії виконання запитів

Для підвищення швидкодії пошукових запитів було обрано індекси GIN [13] та Hash [14]. (див. Додаток А.4)

Оскільки інтернет-магазин потребує часто робити пошук користувача за його ім’ям серед великих об’ємів інформації, було використано індексацію GIN та Hash для прискорення пошуку даних користувача за його контактними даними. У випадку з введенням повного імені, це прискорило запит у 2150 разів. У випадку з пошуком за фрагментом імені, пошуковий запит відбувається в 6 разів швидше. Див додатки А.4.

Отже, очевидно, що використання індексації дозволяє значно прискорити пошукові запити.

Команди SQL, що ініціалізують створення індексів:

|  |
| --- |
| create index on "Client" using hash(name); |

|  |
| --- |
| create index on "Client" using gin(name gin\_trgm\_ops); |

### Опис результатів аналізу предметної галузі

1. *Генерація даних*

В проекті створено 3 способи отримання інформації: парсинг датасетів на основі формату CSV, скрапінг сайту Amazon та псевдо випадкову генерацію.

Парсинг забезпечує записам 3 таблиці, в процесі не виявлено помилок, пов'язаних з форматом файлу. Повторювані дані ігноруються.

Скрапінг забезпечує актуальною інформацією 3 таблиці. Необхідно ввести категорію за якою буде відбуватися пошук.

Генерація інформації була реалізована шляхом транзакцій до серверу БД. Заповнює 3 таблиці, що стосуються замовлень та користувачів, оскільки персональну інформацію неможливо отримати з онлайн сервісів.

1. *Аналіз даних*

За допомогою бібліотек, було забезпечено візуальне представлення чотирьох графіків, що характеризують БД: найпопулярніші 15 категорій продуктів, статистика закупівлі продуктів за всі роки, статистика продажу за всі роки, прибуток магазину за рік. Див додатки А.6.

1. *Валідація та фільтрація даних*

Забезпечена всередині універсальних модулей контроллера та представлення. Пагінація працює за рахунок запитів до БД з атрибутами offset та limit. Валідація працює за допомогою зчитування типу поля з характеристик цього поля, описаних в моделі ORM.

1. *Пошук корисних даних*

Забезпечено створенням контроллера та представлення окремого меню, які містять складні запити до БД, що відповідають потребам бізнес логіки проекту.

### 

### Висновки

В процесі виконання даного курсового проекту було отримано практичні навички роботи з великими масивами даних за допомогою мови програмування Python 3 та СУБД PostgreSQL, їх реплікацію, створення резервних копій та забезпечення отримання цієї інформації різними шляхами.

Було створено програмне забезпечення, яке забезпечило виконання таких пунктів:

1. Генерація рандомізованих даних засобами СУБД.
2. Скрапінг сайтів.
3. Парсинг датасетів.
4. Універсальна робота з моделями ORM.
5. Візуалізація та аналіз даних за допомогою бібліотек Python.
6. Перетворення складних запитів на більш ефективні та компактні за допомогою ORM.
7. Конструктор консольного інтерфейсу користувача.

В ході виконання даного курсового проекту було досягнуто

поставленої мети: було набуто практичних навичок розробки сучасного

програмного забезпечення, що взаємодіє з SQL базами даних, а також

були здобуті навички оформлення відповідного текстового, програмного та

ілюстративного матеріалу у формі проектної документації. У результаті

виконання курсового проекту я навчився писати програмне забезпечення

для SQL баз даних, володіти основами використання СУБД, а також

інструментальними засобами аналізу, візуалізації та роботи з великими обсягами даних.

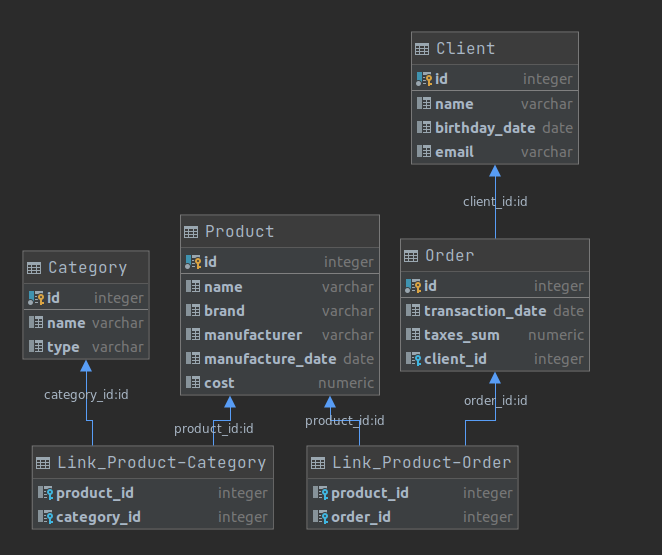
### Література

1. Python [Електронный ресурс]. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Python;
2. NumPy [Електронний ресурс]. – Режим доступу до ресурсу: <https://en.wikipedia.org/wiki/NumPy>;
3. Matplotlib [Електронний ресурс]. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Matplotlib;
4. Pandas [Електронний ресурс]. – Режим доступу до ресурсу: <https://ru.wikipedia.org/wiki/Pandas>;
5. Psycopg2 [Електронний ресурс]. – Режим доступу до ресурсу: [https://www.psycopg.org](https://www.psycopg.org/);
6. Sqlalchemy [Електронний ресурс]. – Режим доступу до ресурсу: https://www.sqlalchemy.org;
7. Фенченко І. utils [Електронний ресурс] / І. Фенченко. – 2020. – Режим доступу до ресурсу: <https://github.com/bujhmt/utils/tree/master/CUI>;
8. Selectorlib [Електронний ресурс]. – Режим доступу до ресурсу: <https://pypi.org/project/selectorlib>;
9. PostgreSQL [Електронний ресурс]. – Режим доступу до ресурсу: [https://www.postgresql.org](https://www.postgresql.org/);
10. PyCharm [Електронний ресурс]. – Режим доступу до ресурсу: <https://www.jetbrains.com/pycharm>;
11. Налаштування реплікації на Ubuntu [Електронний ресурс]. – Режим доступу до ресурсу: [https://www.digitalocean.com/community/tutorials/how-to-set-up-physical-streaming-replication-with-postgresql-12-on-ubuntu-20-04#step-4-%E2%80%94-restarting-and-testing-the-clusters](https://www.digitalocean.com/community/tutorials/how-to-set-up-physical-streaming-replication-with-postgresql-12-on-ubuntu-20-04" \l "step-4-—-restarting-and-testing-the-clusters);
12. Налаштування створення резервних копій БД на Ubuntu [Електронний ресурс]. – Режим доступу до ресурсу: <https://www.postgresqltutorial.com/postgresql-backup-database>;
13. GIN індексація [Електронний ресурс]. – Режим доступу до ресурсу: https://habr.com/en/company/postgrespro/blog/448746;
14. Hash індексація [Електронний ресурс]. – Режим доступу до ресурсу: <https://habr.com/en/company/postgrespro/blog/442776>;

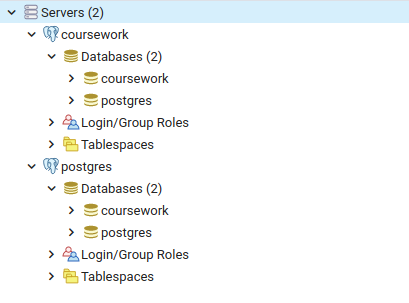
### Додатки

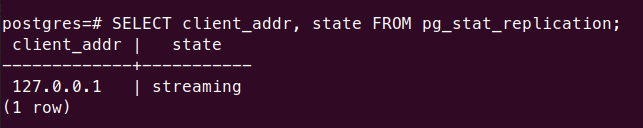
#### Графічні матеріали

1. Структура бази даних:

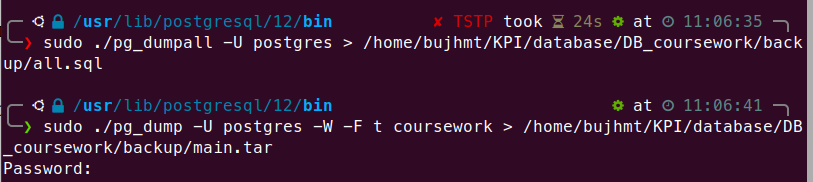


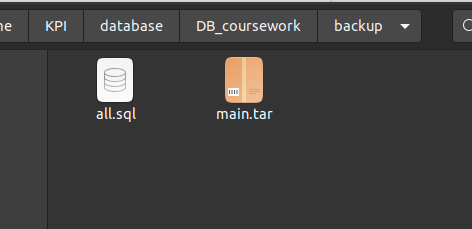
1. Забезпечення засобів реплікації:

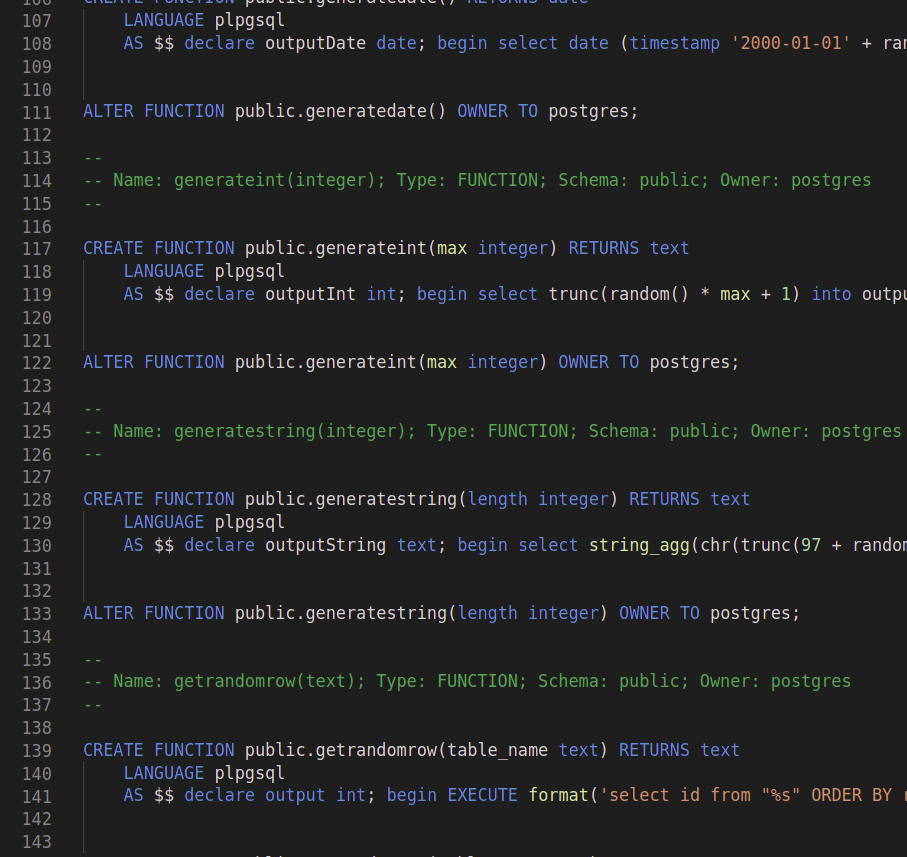




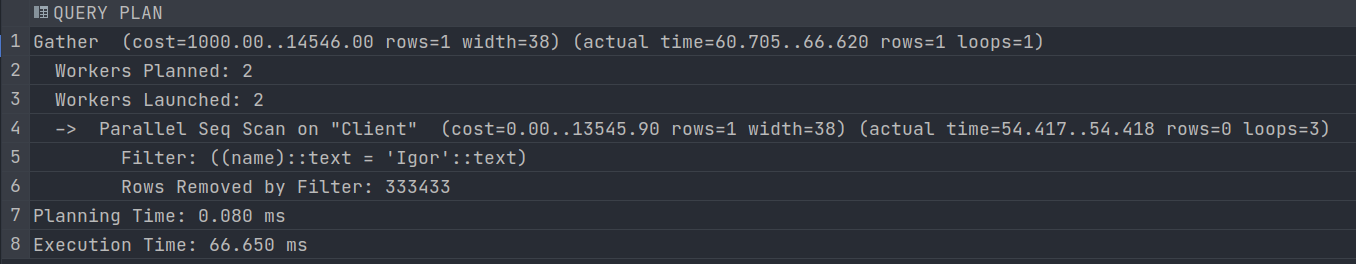
1. Забезпечення засобів резервування / відновлення:

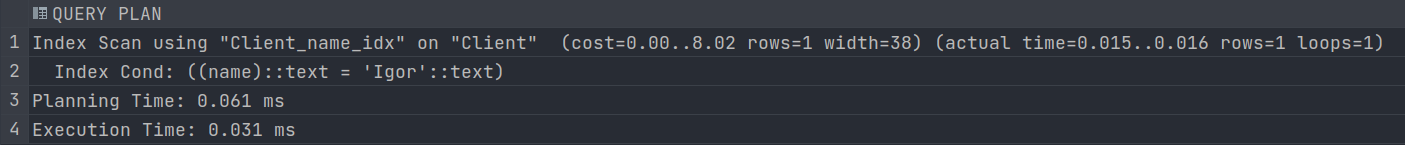


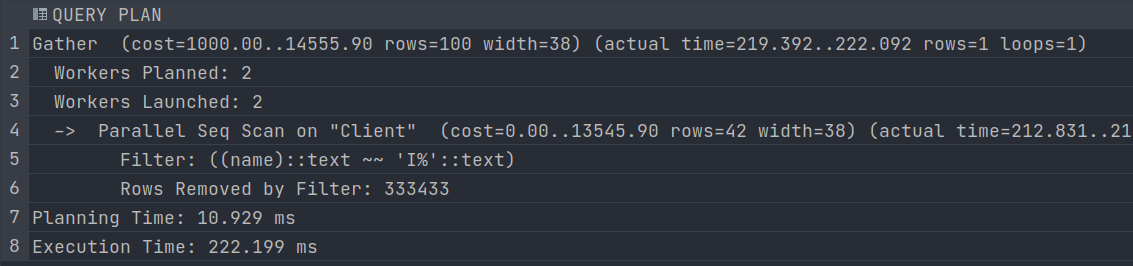


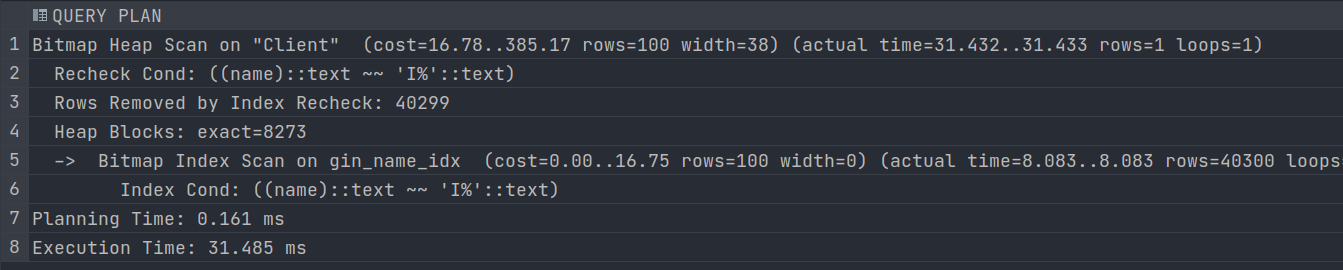


1. Забезпечення засобів підвищення швидкодії виконання запитів:

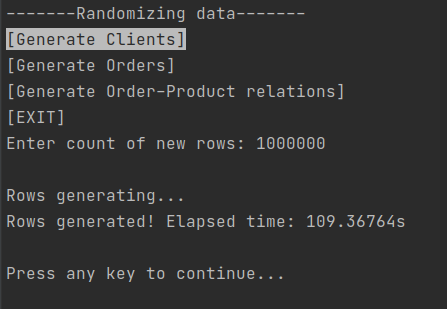


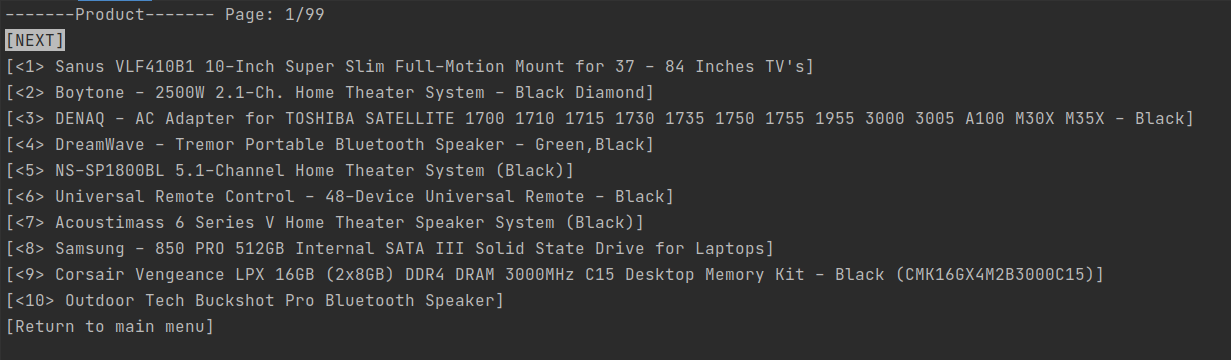


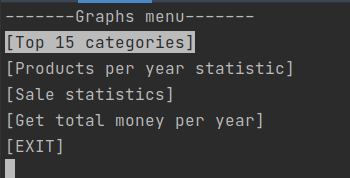


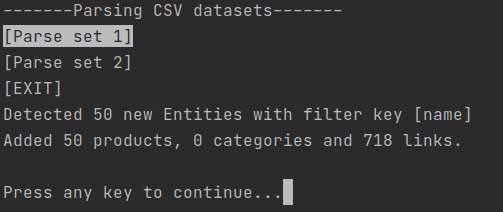


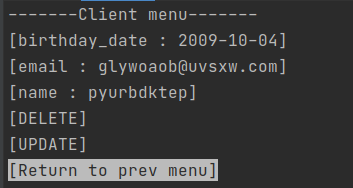
1. Консольний інтерфейс користувача:

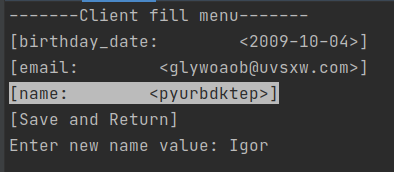




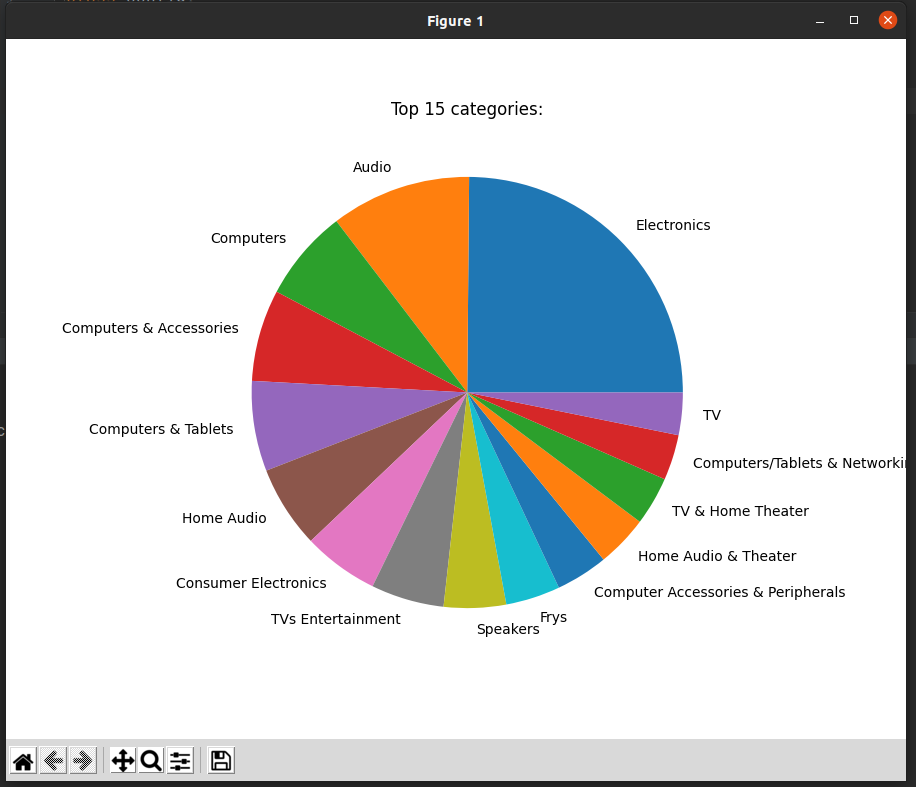


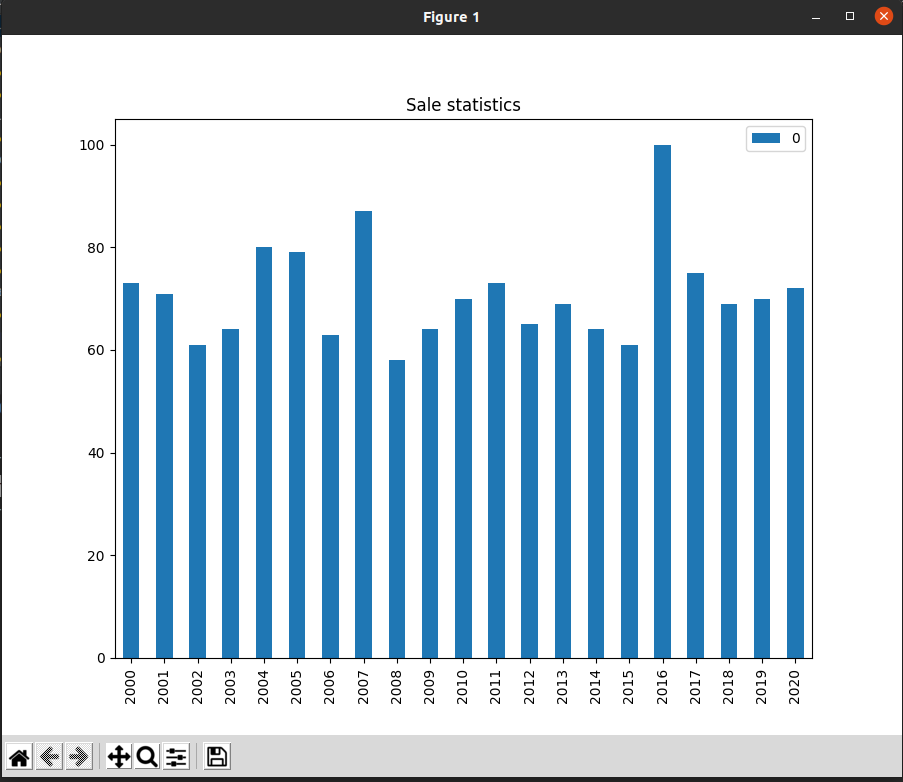


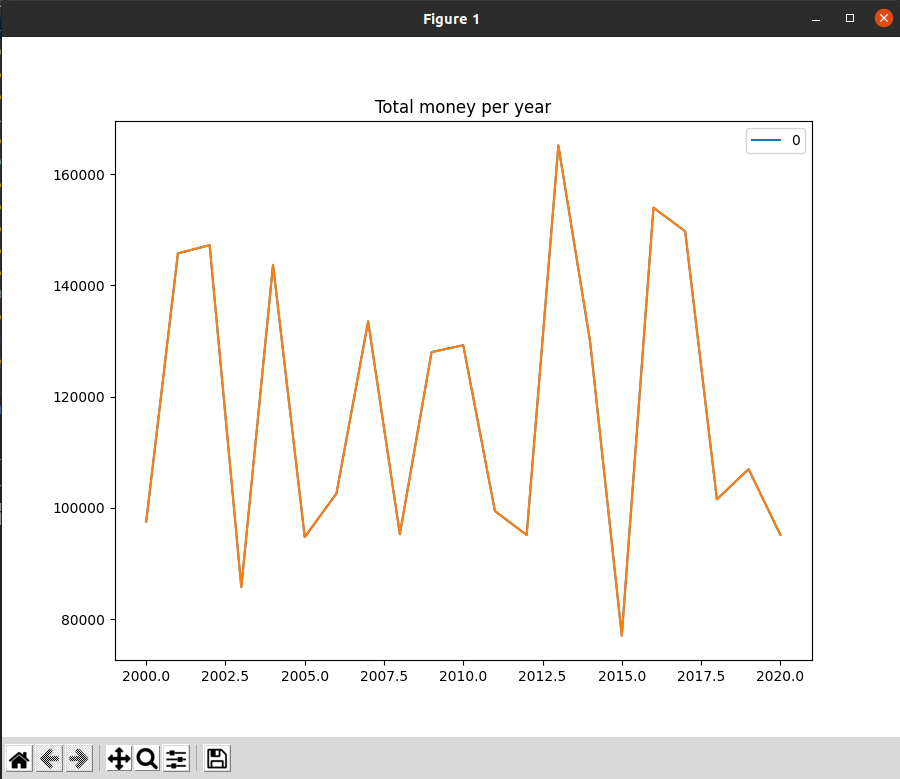




1. Засоби візуалізації та аналізу даних:







#### Фрагменти програмного коду

|  |
| --- |
| **main.py** |
| from db import session, recreate\_database  from views.modelView import EntityView  from CUI.constructor import CUI  from models.client import Client  from models.order import Order  from models.category import Category  from models.product import Product  from views.graphsView import GraphsView  from views.dataView import DataView  from views.searchView import SearchView  if \_\_name\_\_ == '\_\_main\_\_':  #recreate\_database()  cui = CUI('CUI')  cui.addMenu('Models')  cui.addField('Clients', lambda: EntityView(Client).run())  cui.addField('Orders', lambda: EntityView(Order).run())  cui.addField('Categories', lambda: EntityView(Category).run())  cui.addField('Products', lambda: EntityView(Product).run())  cui.finishMenu()  cui.addField('Graphs', lambda: GraphsView().run())  cui.addField('Data', lambda: DataView().run())  cui.addField('Search', lambda: SearchView().run())  cui.run()  session.close() |

|  |
| --- |
| **db.py** |
| from sqlalchemy.ext.declarative import declarative\_base  from sqlalchemy import create\_engine  from sqlalchemy.orm import sessionmaker  from config import Config  Base = declarative\_base()  engine = create\_engine(f'postgresql://{Config.user}:{Config.passwd}@{Config.url}/{Config.db\_name}')  session = sessionmaker(bind=engine)()  def recreate\_database():  print('Recreating database...')  Base.metadata.drop\_all(engine)  Base.metadata.create\_all(engine)  initCustomFunctions()  def defineGenerateStringFunc():  session.execute('create or replace function generateString(length int) '  'returns text '  'language plpgsql '  'as '  '$$ '  'declare '  'outputString text; '  'begin '  'select string\_agg(chr(trunc(97 + random()\*25)::int), \'\') '  'from generate\_series(1, length) '  'into outputString; '  'return outputString; '  'end; '  '$$; ')  session.commit()  def defineGenerateDateFunc():  session.execute('create function generatedate() returns date '  'language plpgsql '  'as '  '$$ '  'declare '  'outputDate date; '  'begin '  'select date (timestamp \'2000-01-01\' + random() \* (timestamp \'2020-12-31\' - timestamp \'2000-01-01\')) '  'into outputDate; '  'return outputDate; '  'end; '  '$$; ')  session.commit()  def defineGenerateIntFunc():  session.execute('create function generateint(max integer) returns text '  'language plpgsql '  'as '  '$$ '  'declare '  'outputInt int; '  'begin '  'select trunc(random() \* max + 1) '  'into outputInt; '  'return outputInt; '  'end; '  '$$; ')  session.commit()  def defineGetRandowRowFunc():  session.execute('create or replace function getrandomrow(table\_name text) returns text '  'language plpgsql '  'as '  '$$ '  'declare '  'output int; '  'begin '  'EXECUTE format(\'select id from "%s" ORDER BY random() LIMIT 1\', table\_name) '  'into output; '  'return output; '  'end; '  '$$; ')  session.commit()  def initCustomFunctions():  defineGenerateStringFunc()  defineGenerateDateFunc()  defineGenerateIntFunc()  defineGetRandowRowFunc() |

|  |
| --- |
| **config.py** |
| class Config:  db\_name = ""  user = ""  passwd = ""  url = "localhost:5432" |

|  |
| --- |
| **graphs.py** |
| import matplotlib.pyplot as plt  import numpy as np  import pandas as pd  from db import session  from models.category import Category  from models.product import Product  from models.order import Order  from sqlalchemy import func, Integer  def getTop15Categories():  results = session.query(Category.name, func.count(Category.name).label('count')) \  .join(Product, Category.Products) \  .group\_by(Category.name) \  .order\_by(func.count('count').desc()) \  .limit(15).all()  listed = list(zip(\*results))  series = pd.Series(np.array(listed[1]), index=listed[0], name='')  series.plot.pie(figsize=(9, 7), title="Top 15 categories:")  plt.plot(series)  plt.show()  def getManufactureDateStat():  results = session.query(func.extract('year', Product.manufacture\_date).cast(Integer).label('year'),  func.count('year')) \  .group\_by('year') \  .order\_by('year').all()  listed = list(zip(\*results))  ts = pd.DataFrame(np.array(listed[1]), listed[0])  ts.plot(kind='bar', figsize=(9, 7), title="Products per year")  plt.plot(ts)  plt.show()  def getTransactionDateStat():  results = session.query(func.extract('year', Order.transaction\_date).cast(Integer).label('year'), func.count('year')) \  .group\_by('year') \  .order\_by('year').all()  listed = list(zip(\*results))  ts = pd.DataFrame(np.array(listed[1]), listed[0])  ts.plot(kind='bar', figsize=(9, 7), title="Sale statistics")  plt.plot(ts)  plt.show()  def getTotalMoneyPerYear():  results = session.query(func.sum(Product.cost).label('sum'),  func.extract('year', Order.transaction\_date).cast(Integer).label('year'))\  .join(Product.Orders)\  .group\_by('year')\  .order\_by('year').all()  listed = list(zip(\*results))  series = pd.DataFrame(np.array([int(num) for num in listed[0]]), index=listed[1])  series.plot(figsize=(9, 7), title="Total money per year")  plt.plot(series)  plt.show() |

|  |
| --- |
| **searchView.py** |
| import math  import time  from controllers.searchController import SearchController  from CUI.constructor import CUI  class SearchView:  def \_\_init\_\_(self):  self.currentMenu = None  self.page = 1  self.per\_page = 10  self.min = None  self.max = None  self.CUI = CUI("Search menu")  self.searchController = SearchController()  self.CUI.addField('Search products cost by cost range', lambda: self.\_\_getProductsByCostRange())  self.CUI.addField('Search all client orders', lambda: self.\_\_getClientOrders())  self.CUI.addField('Search all category products', lambda: self.\_\_getCategoryProducts())  def run(self):  self.CUI.run()  def \_\_changePageParams(self, page: int, per\_page: int):  self.page = page  self.per\_page = per\_page  self.currentMenu.stop()  self.\_\_getProductsByCostRange()  def \_\_exitMenu(self):  self.min = None  self.max = None  self.page = 1  self.per\_page = 10  self.currentMenu.stop()  def \_\_getProductsByCostRange(self):  searchMenu = CUI('Products')  self.currentMenu = searchMenu  try:  if self.min is None or self.max is None:  self.min = int(input('Enter min value: '))  self.max = int(input('Enter max value: '))  if not isinstance(self.min, int) and not isinstance(self.max, int) \  and self.max > 0 and 0 < self.min <= self.max:  raise Exception('Invalid input')  startTime = time.time()  allRecords = self.searchController.getProductsByCostRange(self.min, self.max)  endTime = time.time()  count = math.ceil(len(allRecords) / self.per\_page)  searchMenu.setMsg('\nElapsed time: ' + str(endTime - startTime)[:9] + 's'  '\nPage: ' + str(self.page) + '/' + str(count))  if self.page < math.floor(count):  searchMenu.addField('NEXT', lambda: self.\_\_changePageParams(self.page + 1, self.per\_page))  if self.page > 1:  searchMenu.addField('PREV', lambda: self.\_\_changePageParams(self.page - 1, self.per\_page))  searchMenu.addField('<ID> | Product name | Product cost')  for record in self.searchController.getProductsByCostRange(self.min, self.max, self.page, self.per\_page):  searchMenu.addField(f"<{record[0]}> | {record[1]} | {record[2]}")  except Exception as err:  searchMenu.setMsg(str(err))  searchMenu.addField('Return to prev menu', lambda: self.\_\_exitMenu())  searchMenu.run(False)  def \_\_getClientOrders(self):  searchMenu = CUI('Client Orders')  self.currentMenu = searchMenu  try:  client\_id = int(input('Enter client\_id: '))  if not (isinstance(client\_id, int) and client\_id > 0):  raise Exception('Invalid input')  startTime = time.time()  allRecords = self.searchController.getAllClientOrders(client\_id)  endTime = time.time()  searchMenu.setMsg('\nElapsed time: ' + str(endTime - startTime)[:9] + 's'  '\nRows num: ' + str(len(allRecords)))  searchMenu.addField('<Order id> | transaction date | taxes\_sum')  for record in allRecords:  searchMenu.addField(f"<{record[0]}> | {record[1]} | {record[2]}")  except Exception as err:  searchMenu.setMsg(str(err))  searchMenu.run('Return to prev menu')  def \_\_getCategoryProducts(self):  searchMenu = CUI('Product categories')  self.currentMenu = searchMenu  try:  product\_id = int(input('Enter product id: '))  if not (isinstance(product\_id, int) and product\_id > 0):  raise Exception('Invalid input')  startTime = time.time()  allRecords = self.searchController.getAllProductCategories(product\_id)  endTime = time.time()  searchMenu.setMsg('\nElapsed time: ' + str(endTime - startTime)[:9] + 's'  '\nRows num: ' + str(len(allRecords)))  searchMenu.addField(f'Product id: {allRecords[0][0]} | Product name: {allRecords[0][1][0]}\nCategories:')  for record in allRecords[0][2]:  searchMenu.addField(record)  except Exception as err:  searchMenu.setMsg(str(err))  searchMenu.run('Return to prev menu') |

|  |
| --- |
| **modelView.py** |
| import math  from controllers.modelController import ModelController  from CUI.constructor import CUI  exec\_bad\_chars = set('{}()[],;\\/\'\"')  class EntityView:  def \_\_init\_\_(self, instance):  self.instance = instance  self.page = 1  self.itemsCurrentMenu = [None, None]  self.per\_page = 10  self.CUI = CUI(f"{self.instance.\_\_name\_\_} model menu")  self.Controller = ModelController(instance)  self.CUI.addField(f'Add {self.instance.\_\_name\_\_}', lambda: self.\_\_addItem())  self.CUI.addField(f'{self.instance.\_\_name\_\_}', lambda: self.\_\_getItems())  def \_\_supportFillItemFunc(self, key, value, item):  try:  new\_value = input(f'Enter new {key} value: ')  if isinstance(new\_value, str) and len(new\_value) > 0:  if new\_value.isdigit():  setattr(item, key, int(new\_value))  else:  setattr(item, key, new\_value)  else:  raise Exception('Incorrect input')  self.currentMenu.renameField(f'{key}: <{value}>', f'{key}: <{new\_value}>')  except Exception as err:  self.currentMenu.setMsg(str(err))  def \_\_supportFillFunc(self, key, mapped):  try:  value = input(f'Enter new {key} value: ')  old\_value = None  if key in mapped and mapped[key] is not None: old\_value = mapped[key]  if isinstance(value, str) and len(value) > 0:  if value.isdigit():  mapped[key] = int(value)  else:  mapped[key] = value  else:  raise Exception('Incorrect input')  if old\_value is None:  self.currentMenu.renameField(f'{key}', f'{key}: <{value}>')  else:  self.currentMenu.renameField(f'{key}: <{old\_value}>', f'{key}: <{value}>')  except Exception as err:  self.currentMenu.setMsg(str(err))  def \_\_fillEntityMenu(self, \*args):  self.currentMenu = CUI(f'{self.instance.\_\_name\_\_} fill menu')  try:  if len(args) > 0 and isinstance(args[0], self.instance):  item = args[0]  for (key, value) in self.Controller.getModelEntityMappedKeys(item).items():  self.currentMenu.addField(f'{key}: <{value}>',  lambda key=key, value=value: self.\_\_supportFillItemFunc(key, value, item))  elif len(args) > 0 and isinstance(args[0], dict):  mapped = args[0]  for key in self.Controller.getModelKeys():  mapped[key] = None  self.currentMenu.addField(f'{key}', lambda key=key: self.\_\_supportFillFunc(key, mapped))  else:  raise Exception('Invalid arguments')  except Exception as err:  self.currentMenu.setMsg(str(err))  self.currentMenu.run('Save and Return')  def run(self):  self.CUI.run()  def \_\_addItem(self):  try:  mapped = {}  self.\_\_fillEntityMenu(mapped)  exec\_str = ''  for value in mapped.values():  if value is None or \  (isinstance(value, str) and  any((char in exec\_bad\_chars) for char in value)) \  : raise Exception('Invalid entity fill')  if isinstance(value, str):  exec\_str += f"'{value}', "  else: exec\_str += f"{value}, "  exec("self.entity = self.instance(%s)" % exec\_str[:-1])  entityId = self.Controller.add(self.entity)  self.CUI.setMsg(f'New Entity created! Id: {entityId}')  except Exception as err:  self.CUI.setMsg(str(err))  def \_\_changePageParams(self, page: int, per\_page: int):  self.page = page  self.per\_page = per\_page  self.itemsCurrentMenu[0].stop()  self.\_\_getItems()  def \_\_getItems(self):  itemsMenu = CUI(self.instance.\_\_name\_\_)  self.itemsCurrentMenu[0] = itemsMenu  try:  count = math.ceil(self.Controller.getCount() / self.per\_page)  if self.page < count:  itemsMenu.addField('NEXT', lambda: self.\_\_changePageParams(self.page + 1, self.per\_page))  if self.page > 1:  itemsMenu.addField('PREV', lambda: self.\_\_changePageParams(self.page - 1, self.per\_page))  entities = self.Controller.getRange(self.page, self.per\_page)  itemsMenu.setMsg(f' Page: {self.page}/{count}')  for entity in entities:  if 'name' in self.Controller.getModelKeys():  itemsMenu.addField(f"<{entity.id}> {entity.name}", lambda id=entity.id: self.\_\_getItem(id))  else:  itemsMenu.addField(f"<{entity.id}>", lambda id=entity.id: self.\_\_getItem(id))  except Exception as err:  itemsMenu.setMsg(str(err))  itemsMenu.run('Return to main menu')  def \_\_updateItem(self, item):  self.\_\_fillEntityMenu(item)  self.Controller.update(item)  self.itemsCurrentMenu[1].stop()  self.\_\_getItem(item.id)  def \_\_deleteItem(self, id: int):  self.Controller.delete(id)  self.itemsCurrentMenu[1].stop()  self.\_\_supportCUIFunc()  def \_\_supportCUIFunc(self):  self.itemsCurrentMenu[1].stop()  self.\_\_changePageParams(self.page, self.per\_page)  def \_\_getItem(self, id: int):  itemMenu = CUI(f'{self.instance.\_\_name\_\_} menu')  self.itemsCurrentMenu[1] = itemMenu  try:  item = self.Controller.getById(id)  for (key, value) in self.Controller.getModelEntityMappedKeys(item).items():  itemMenu.addField(str(key) + ' : ' + str(value))  itemMenu.addField('DELETE', lambda: self.\_\_deleteItem(item.id))  itemMenu.addField('UPDATE', lambda: self.\_\_updateItem(item))  itemMenu.addField('Return to prev menu', lambda: self.\_\_supportCUIFunc())  except Exception as err:  itemMenu.setMsg(str(err))  itemMenu.run(False) |

|  |
| --- |
| **graphsView.py** |
| from CUI.constructor import CUI  from visualzation.graphs import getTop15Categories,\  getManufactureDateStat,\  getTransactionDateStat,\  getTotalMoneyPerYear  class GraphsView(object):  def \_\_init\_\_(self):  self.CUI = CUI('Graphs menu')  self.CUI.addField('Top 15 categories', lambda: getTop15Categories())  self.CUI.addField('Products per year statistic', lambda: getManufactureDateStat())  self.CUI.addField('Sale statistics', lambda: getTransactionDateStat())  self.CUI.addField('Get total money per year', lambda: getTotalMoneyPerYear())  def run(self):  self.CUI.run() |

|  |
| --- |
| **dataView.py** |
| from CUI.constructor import CUI  from parsingData.parseCsv import parse1, parse2  from scrapping.amazon import getProductsByCategory  from randomizing.generateData import generateOrders, generateClients, generateOrderRelation  class DataView(object):  def \_\_init\_\_(self):  self.CUI = CUI('New data menu')  self.CUI.addField('Scrapping', lambda: self.\_\_scrapping())  self.CUI.addField('Parsing', lambda: self.\_\_parsingDataSets())  self.CUI.addField('Randomizing', lambda: self.\_\_randomizing())  def run(self):  self.CUI.run()  def \_\_scrapping(self):  localMenu = CUI('Scrapping data')  try:  localMenu.addField('Get data from Amazon', lambda: getProductsByCategory())  except Exception as err:  localMenu.setMsg(str(err))  localMenu.run()  def \_\_parsingDataSets(self):  localMenu = CUI('Parsing CSV datasets')  try:  localMenu.addField('Parse set 1', lambda: parse1())  localMenu.addField('Parse set 2', lambda: parse2())  except Exception as err:  localMenu.setMsg(str(err))  localMenu.run()  def \_\_randomizing(self):  localMenu = CUI('Randomizing data')  try:  localMenu.addField('Generate Clients', lambda: generateClients())  localMenu.addField('Generate Orders', lambda: generateOrders())  localMenu.addField('Generate Order-Product relations', lambda: generateOrderRelation())  except Exception as err:  localMenu.setMsg(str(err))  localMenu.run() |

|  |
| --- |
| **amazon.py** |
| from selectorlib import Extractor  import requests  import random  import re  from datetime import datetime  from models.category import Category  from models.product import Product  from db import session  from models.links import links\_products\_categories  e = Extractor.from\_yaml\_file('./scrapping/search\_results.yml')  def scrape(url):  headers = {  'dnt': '1',  'upgrade-insecure-requests': '1',  'user-agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_15\_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/83.0.4103.61 Safari/537.36',  'accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,\*/\*;q=0.8,application/signed-exchange;v=b3;q=0.9',  'sec-fetch-site': 'same-origin',  'sec-fetch-mode': 'navigate',  'sec-fetch-user': '?1',  'sec-fetch-dest': 'document',  'referer': 'https://www.amazon.com/',  'accept-language': 'en-GB,en-US;q=0.9,en;q=0.8',  }  print("Downloading %s"%url)  r = requests.get(url, headers=headers)  if r.status\_code > 500:  if "To discuss automated access to Amazon data please contact" in r.text:  print("Page %s was blocked by Amazon. Please try using better proxies\n"%url)  else:  print("Page %s must have been blocked by Amazon as the status code was %d"%(url,r.status\_code))  return None  return e.extract(r.text)  def getProductsByCategory():  category = input('Enter category for scrapping: ')  data = scrape(f'https://www.amazon.com/s?k={category}')  if data:  products\_count = 0  categories\_count = 0  links\_count = 0  for raw in data['products']:  print(f"Get product from {raw['url']}")  if raw['price'] != None: price = int(float(re.findall("\d+\.\d+", raw['price'])[0]))  else: price = random.randint(200, 5000)  new\_product = Product(raw['title'],  '',  '',  datetime.today().strftime('%Y-%m-%d'),  price)  session.add(new\_product)  session.commit()  products\_count += 1  session.refresh(new\_product)  # Adding categories to database:  checked\_categories = session.query(Category).filter(Category.name == category).all()  if len(checked\_categories) == 0:  new\_category = Category(category, category)  session.add(new\_category)  session.commit()  categories\_count += 1  session.refresh(new\_category)  # Adding link:  ins = links\_products\_categories.insert().values(product\_id=new\_product.id,  category\_id=new\_category.id)  session.execute(ins)  else:  ins = links\_products\_categories.insert().values(product\_id=new\_product.id,  category\_id=checked\_categories[0].id)  session.execute(ins)  links\_count += 1  session.commit()  print(f'\nAdded {products\_count} products, {categories\_count} categories and {links\_count} links.')  input('\nPress any key to continue...') |

|  |
| --- |
| **generateData.py** |
| from db import session  import time  from models.client import Client  from models.order import Order  from models.product import Product  def inputCount():  count = input("Enter count of new rows: ")  if not count.isdigit() or int(count) < 1: raise Exception('Incorrect input')  return int(count)  def generateClients(\*args):  count = None  if len(args) == 0:  count = inputCount()  else: count = args[0]  startTime = time.time()  if len(args) == 0: print('\nRows generating...')  session.execute(f"INSERT INTO \"Client\" (name, birthday\_date, email) "  f"SELECT generatestring(10), "  f"generatedate(), "  f"concat(generatestring(8), '@', generatestring(5), '.com') "  f"from generate\_series(1, {count})")  session.commit()  endTime = time.time()  if len(args) == 0:  print('Rows generated! Elapsed time: ' + str(endTime - startTime)[:9] + 's')  input('\nPress any key to continue...')  def generateOrders(\*args):  count = None  if len(args) == 0:  count = inputCount()  else:  count = args[0]  clientsCount = session.query(Client).count()  startTime = time.time()  if len(args) == 0: print('\nRows generating...')  if int(count) / 5 > clientsCount:  generateClients(int(int(count) / 5) - clientsCount)  session.execute(f"INSERT INTO \"Order\" (taxes\_sum, transaction\_date, client\_id) "  f"SELECT generateint(500)::numeric, "  f"generatedate(), "  f"getrandomrow('Client')::int "  f"FROM generate\_series(1, {count})")  session.commit()  endTime = time.time()  if len(args) == 0:  print('Rows generated! Elapsed time: ' + str(endTime - startTime)[:9] + 's')  input('\nPress any key to continue...')  def generateOrderRelation(\*args):  count = None  if len(args) == 0:  count = inputCount()  else:  count = args[0]  if session.query(Product).count() < 10:  raise Exception('Too few product in the database. Please scrap or parse some data')  ordersCount = session.query(Order).count()  startTime = time.time()  if len(args) == 0: print('\nRows generating...')  if int(count) / 10 > ordersCount:  generateOrders(int(int(count) / 10) - ordersCount)  session.execute(f"INSERT INTO \"Link\_Product-Order\" (product\_id, order\_id) "  f"SELECT getrandomrow('Product')::int, "  f"getrandomrow('Order')::int "  f"FROM generate\_series(1,{count})")  session.commit()  endTime = time.time()  if len(args) == 0:  print('Rows generated! Elapsed time: ' + str(endTime - startTime)[:9] + 's')  input('\nPress any key to continue...') |

|  |
| --- |
| **parseCsv.py** |
| import csv  import random  from db import session  from models.category import Category  from models.product import Product  from models.links import links\_products\_categories  def parseCsv(path: str, keys: [], filterKey):  with open(path, 'r', encoding='Latin1') as csv\_data:  reader = csv.DictReader(csv\_data)  entities = []  prev\_product = ''  for row in reader:  if prev\_product != row[filterKey]:  entities.append(row)  if len(entities) % 100 == 0:  print(f'Detected {str(len(entities))} new Entities with filter key [{filterKey}]')  prev\_product = row[filterKey]  print(f'Detected {str(len(entities))} new Entities with filter key [{filterKey}]')  filteredEntities = []  for entity in entities:  item = {}  for key in keys:  item[key] = entity[key]  filteredEntities.append(item)  return filteredEntities  def parse1():  data\_dict = parseCsv('../data/csv/DatafinitiElectronicsProductData.csv',  ['name', 'brand', 'categories', 'dateAdded', 'manufacturer', 'primaryCategories'],  'name')  # Adding Products to database:  products\_count = 0  categories\_count = 0  links\_count = 0  for row in data\_dict:  new\_product = Product(row['name'],  row['brand'],  row['manufacturer'],  row['dateAdded'],  random.randint(200, 5000))  session.add(new\_product)  session.commit()  products\_count += 1  session.refresh(new\_product)  # Adding categories to database:  category\_names = row['categories'].split(',')  for category\_name in category\_names:  checked\_categories = session.query(Category).filter(Category.name == category\_name).all()  if len(checked\_categories) == 0:  new\_category = Category(category\_name, row['primaryCategories'])  session.add(new\_category)  session.commit()  categories\_count += 1  session.refresh(new\_category)  #Adding link:  ins = links\_products\_categories.insert().values(product\_id=new\_product.id,  category\_id=new\_category.id)  session.execute(ins)  else:  ins = links\_products\_categories.insert().values(product\_id=new\_product.id,  category\_id=checked\_categories[0].id)  session.execute(ins)  links\_count += 1  session.commit()  print(f'Added {products\_count} products, {categories\_count} categories and {links\_count} links.')  input('\nPress any key to continue...')  def parse2():  data\_dict = parseCsv('../data/csv/DatafinitiElectronicsProductsPricingData.csv',  ['name', 'brand', 'categories', 'dateAdded', 'manufacturer', 'primaryCategories', 'prices.amountMin'],  'name')  # Adding Products to database:  products\_count = 0  categories\_count = 0  links\_count = 0  for row in data\_dict:  new\_product = Product(row['name'],  row['brand'],  row['manufacturer'],  row['dateAdded'],  int(float(row['prices.amountMin'])))  session.add(new\_product)  session.commit()  products\_count += 1  session.refresh(new\_product)  # Adding categories to database:  category\_names = row['categories'].split(',')  for category\_name in category\_names:  checked\_categories = session.query(Category).filter(Category.name == category\_name).all()  if len(checked\_categories) == 0:  new\_category = Category(category\_name, row['primaryCategories'])  session.add(new\_category)  session.commit()  categories\_count += 1  session.refresh(new\_category)  # Adding link:  ins = links\_products\_categories.insert().values(product\_id=new\_product.id,  category\_id=new\_category.id)  session.execute(ins)  else:  ins = links\_products\_categories.insert().values(product\_id=new\_product.id,  category\_id=checked\_categories[0].id)  session.execute(ins)  links\_count += 1  session.commit()  print(  f'Added {products\_count} products, {categories\_count} categories and {links\_count} links.')  input('\nPress any key to continue...') |

|  |
| --- |
| **product.py** |
| from sqlalchemy import Column, Integer, Numeric, String, Date, func  from sqlalchemy.orm import relationship  from db import Base  from models.links import links\_products\_categories, links\_products\_orders  class Product(Base):  \_\_tablename\_\_ = 'Product'  id = Column(Integer, primary\_key=True)  name = Column(String)  brand = Column(String)  manufacturer = Column(String)  manufacture\_date = Column(Date, default=func.now())  cost = Column(Numeric)  Orders = relationship("Order", secondary=links\_products\_orders, cascade="all, delete")  Categories = relationship("Category", secondary=links\_products\_categories, cascade="all, delete")  def \_\_repr\_\_(self):  return "<Product(name='%s'," \  " brand='%s'," \  " manufacturer='%s'," \  " manufacture\_date='%s'," \  " cost=%i, " % \  (self.name,  self.brand,  self.manufacturer,  self.manufacture\_date,  self.cost)  def \_\_init\_\_(self,  name: str,  brand: str,  manufacturer: str,  manufacture\_date: str,  cost: int):  self.name = name  self.brand = brand  self.manufacturer = manufacturer  self.manufacture\_date = manufacture\_date  self.cost = cost |

|  |
| --- |
| **order.py** |
| from sqlalchemy import Column, Integer, Numeric, Date, func, ForeignKey  from sqlalchemy.orm import relationship, backref  from db import Base  from models.links import links\_products\_orders  class Order(Base):  \_\_tablename\_\_ = 'Order'  id = Column(Integer, primary\_key=True)  transaction\_date = Column(Date, default=func.now())  taxes\_sum = Column(Numeric)  client\_id = Column(Integer, ForeignKey('Client.id', ondelete='CASCADE'))  Client = relationship("Client", backref=backref("Order", uselist=False, cascade="all,delete"))  Products = relationship("Product", secondary=links\_products\_orders, cascade="all, delete")  def \_\_repr\_\_(self):  return "<Order(taxes\_sum=%i, transaction\_date='%s', client\_id=%i)>" % \  (self.taxes\_sum, self.transaction\_date, self.client\_id)  def \_\_init\_\_(self, transaction\_date: str, taxes\_sum: int, client\_id: int):  self.taxes\_sum = taxes\_sum  self.transaction\_date = transaction\_date  self.client\_id = client\_id |

|  |
| --- |
| **links.py** |
| from sqlalchemy import Column, Integer, Table, ForeignKey  from db import Base  links\_products\_orders = Table(  'Link\_Product-Order', Base.metadata,  Column('product\_id', Integer, ForeignKey('Product.id', ondelete='CASCADE')),  Column('order\_id', Integer, ForeignKey('Order.id', ondelete='CASCADE'))  )  links\_products\_categories = Table(  'Link\_Product-Category', Base.metadata,  Column('product\_id', Integer, ForeignKey('Product.id', ondelete='CASCADE')),  Column('category\_id', Integer, ForeignKey('Category.id', ondelete='CASCADE'))  ) |

|  |
| --- |
| **client.py** |
| from sqlalchemy import Column, Integer, String, Date, func  from db import Base  class Client(Base):  \_\_tablename\_\_ = 'Client'  id = Column(Integer, primary\_key=True)  name = Column(String)  birthday\_date = Column(Date, default=func.now())  email = Column(String)  def \_\_repr\_\_(self):  return "<Client(name='%s', birthday\_date='%s', email='%s')>" % \  (self.name, self.birthday\_date, self.email)  def \_\_init\_\_(self, name: str, birthday\_date: str, email: str):  self.name = name  self.birthday\_date = birthday\_date  self.email = email |

|  |
| --- |
| **category.py** |
| from sqlalchemy import Column, Integer, String  from db import Base  from sqlalchemy.orm import relationship, backref  from models.links import links\_products\_categories  class Category(Base):  \_\_tablename\_\_ = 'Category'  id = Column(Integer, primary\_key=True)  name = Column(String)  type = Column(String)  Products = relationship("Product", secondary=links\_products\_categories, cascade="all, delete")  def \_\_repr\_\_(self):  return "<Category(name='%s', type='%s')>" % \  (self.name, self.type)  def \_\_init\_\_(self, name: str, type: str):  self.name = name  self.type = type |

|  |
| --- |
| **constructor.py** |
| import os  import sys  # console utils:  def printBold(str):  BOLD = '\33[7m'  CEND = '\033[0m'  print(BOLD + str + CEND)  def clear():  os.system('cls' if os.name == 'nt' else 'clear')  def getch():  if os.name == 'nt':  import msvcrt  return msvcrt.getch()  else:  import tty  import termios  fd = sys.stdin.fileno()  old\_settings = termios.tcgetattr(fd)  try:  tty.setraw(sys.stdin.fileno())  ch = sys.stdin.read(1)  finally:  termios.tcsetattr(fd, termios.TCSADRAIN, old\_settings)  return ch  # Tree Node  class Node(object):  def \_\_init\_\_(self, title, on\_press):  self.title = title  self.on\_press = on\_press  self.childs = []  self.root = None  def append(self, \*args):  try:  if len(args) == 1 and isinstance(args[0], Node):  args[0].root = self  self.childs.append(args[0])  if len(args) == 2 and isinstance(args[0], str):  newNode = Node(args[0], args[1])  newNode.root = self  self.childs.append(newNode)  except Exception as err:  print("Error! ", err)  # CUI class  class CUI(object):  def \_\_setBreakStatus(self, status: bool):  self.\_\_isBreakON = status  def \_\_init\_\_(self, mainMenuTitle='Main menu'):  self.root = Node(mainMenuTitle, lambda: 0)  self.\_\_currentNode = self.root  # support:  self.\_\_BREAK\_NODE = Node("EXIT", lambda: self.\_\_setBreakStatus(False))  self.\_\_EMPTY\_NODE = Node("", lambda: 0)  # private fields:  self.\_\_currentPos = 1  self.\_\_isBreakON = True  self.\_\_msg = ''  def \_\_print(self):  clear()  # custom items  print(f'-------{self.\_\_currentNode.title}-------' + self.\_\_msg)  for i in range(len(self.\_\_currentNode.childs)):  if i == self.\_\_currentPos - 1:  printBold(f'[{self.\_\_currentNode.childs[i].title}]')  else:  print(f'[{self.\_\_currentNode.childs[i].title}]')  def \_\_inputController(self):  return getch()  def \_\_stepController(self, char):  upperLimit: int = len(self.\_\_currentNode.childs)  charCode: int = ord(char.lower())  if (charCode == 119 or charCode == 97) and self.\_\_currentPos > 1: self.\_\_currentPos += -1  if (charCode == 115 or charCode == 98) and self.\_\_currentPos < upperLimit: self.\_\_currentPos += 1  try:  if charCode == 10 or charCode == 13: self.\_\_currentNode.childs[self.\_\_currentPos - 1].on\_press()  except Exception as err:  self.setMsg(' Invalid operation! ' + str(err)[0:70])  def \_\_goToCurrentNode(self):  self.\_\_currentNode = self.\_\_currentNode.childs[self.\_\_currentPos - 1]  self.\_\_currentPos = 1  def \_\_goToParent(self):  self.\_\_currentNode = self.\_\_currentNode.root  self.\_\_currentPos = 1  # public fields:  def run(self, \*args):  self.\_\_currentNode = self.root  exit\_str = "EXIT"  if len(args) > 0 and isinstance(args[0], str): exit\_str = args[0]  if not (len(args) > 0 and isinstance(args[0], bool) and args[0] is False):  self.\_\_currentNode.append(exit\_str, lambda: self.\_\_setBreakStatus(False))  while (self.\_\_isBreakON):  self.\_\_print()  self.\_\_stepController(self.\_\_inputController())  clear()  def addField(self, title, \*args):  if len(args) > 0:  self.\_\_currentNode.append(title, args[0])  else:  self.\_\_currentNode.append(title, lambda: 0)  def addMenu(self, title):  newNode = Node(title, lambda: self.\_\_goToCurrentNode())  newNode.append("Return to prev Menu", lambda: self.\_\_goToParent())  self.\_\_currentNode.append(newNode)  self.\_\_currentNode = newNode  def finishMenu(self):  if self.\_\_currentNode.root != None:  self.\_\_goToParent()  def renameField(self, current: str, new: str):  try:  if len(current) > 0 and len(new) > 0:  if self.\_\_currentNode.title == current:  self.\_\_currentNode.title = new  for i in range(len(self.\_\_currentNode.childs)):  if self.\_\_currentNode.childs[i].title == current:  self.\_\_currentNode.childs[i].title = new  else:  raise Exception('Invalid title')  except Exception as err:  print("Error! ", err)  def deleteField(self, title: str):  try:  if len(title) > 0:  for i in range(len(self.\_\_currentNode.childs)):  if self.\_\_currentNode.childs[i].title == title:  del self.\_\_currentNode.childs[i]  else:  raise Exception('Invalid title')  except Exception as err:  print("Error! ", err)  def setMsg(self, msg: str):  if len(msg) > 105:  msg = msg[0:msg.find('\n')]  self.\_\_msg = msg  def stop(self):  self.\_\_setBreakStatus(False) |

|  |
| --- |
| **searchController.py** |
| from db import session  from models.product import Product  from models.order import Order  from models.category import Category  from sqlalchemy import func  class SearchController(object):  def getProductsByCostRange(self, min: int, max: int, \*args):  try:  if len(args) == 0:  return session.query(Product.id, Product.name, Product.cost) \  .filter(Product.cost > min, Product.cost < max) \  .order\_by(Product.cost).all()  else:  return session.query(Product.id, Product.name, Product.cost) \  .filter(Product.cost > min, Product.cost < max) \  .order\_by(Product.cost)\  .limit(args[1])\  .offset(args[0] \* args[1]).all()  except Exception as err:  print("Get error! ", err)  def getAllClientOrders(self, client\_id: int):  try:  return session.query(Order.id, Order.transaction\_date, Order.taxes\_sum)\  .filter(Order.client\_id == client\_id)\  .order\_by(Order.transaction\_date).all()  except Exception as err:  raise str(err)  def getAllProductCategories(self, product\_id: int):  try:  return session.query(Product.id, func.array\_agg(Product.name), func.array\_agg(Category.name))\  .join(Product.Categories)\  .filter(Product.id == product\_id)\  .group\_by(Product.id).all()  except Exception as err:  raise str(err) |

|  |
| --- |
| **modelController.py** |
| from sqlalchemy import func, select  from sqlalchemy.orm.attributes import InstrumentedAttribute  from db import session  class ModelController(object):  def \_\_init\_\_(self, instance):  self.instance = instance  def getRange(self, page: int, per\_page: int):  items = []  try:  page -= 1  items = session.query(self.instance) \  .order\_by(self.instance.id.asc()) \  .offset(page \* per\_page) \  .limit(per\_page) \  .all()  except Exception as err:  print("Get error! ", err)  session.rollback()  raise err  return items  def add(self, item):  try:  if not isinstance(item, self.instance):  raise Exception('Invalid arguments')  session.add(item)  session.commit()  session.refresh(item)  return item.id  except Exception as err:  print("Add error! ", err)  exit(1)  session.rollback()  raise err  def getById(self, itemId):  try:  return session.query(self.instance).get(itemId)  except Exception as err:  print("Get by id error! ", err)  session.rollback()  raise err  def delete(self, itemId):  try:  deletedItem = self.getById(itemId)  if deletedItem is None: raise Exception('This id doesn\'t exists')  session.query(self.instance).filter(self.instance.id == itemId).delete()  session.commit()  return deletedItem  except Exception as err:  print("Delete error! ", err)  session.rollback()  raise err  def update(self, item):  try:  if not isinstance(item, self.instance):  raise Exception('Invalid arguments')  session.query(self.instance) \  .filter(self.instance.id == item.id) \  .update(self.getModelEntityMappedKeys(item))  session.commit()  return True  except Exception as err:  print("Update error! ", err)  session.rollback()  raise err  def getCount(self):  try:  return session.execute(select([func.count()]).select\_from(self.instance)).scalar()  except Exception as err:  print("Get count error! ", err)  session.rollback()  raise err  def getModelKeys(self):  keys = []  for entity in self.instance.\_\_dict\_\_.items():  key = entity[0]  key\_type = entity[1]  if type(key\_type) is InstrumentedAttribute and key is not 'id' and not key[0].isupper():  keys.append(key)  return keys  def getModelEntityMappedKeys(self, item):  mapped\_values = {}  for entity in item.\_\_dict\_\_.items():  key = entity[0]  value = entity[1]  if key is not '\_sa\_instance\_state' and key is not 'id':  mapped\_values[key] = value  return mapped\_values |