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 і засоби управління»**

Тема: «Ознайомлення з базовими операціями СУБД PostgreSQL»

Виконав: студент III курсу

ФПМ групи КВ-83

Бусленко Віталій

Перевірив: Павловський В. І.

Київ – 2020

*Мета роботи:* здобуття практичних навичок проектування та побудови реляційних баз даних та створення прикладних програм з базами даних.

*Загальне завдання роботи полягає у наступному*:

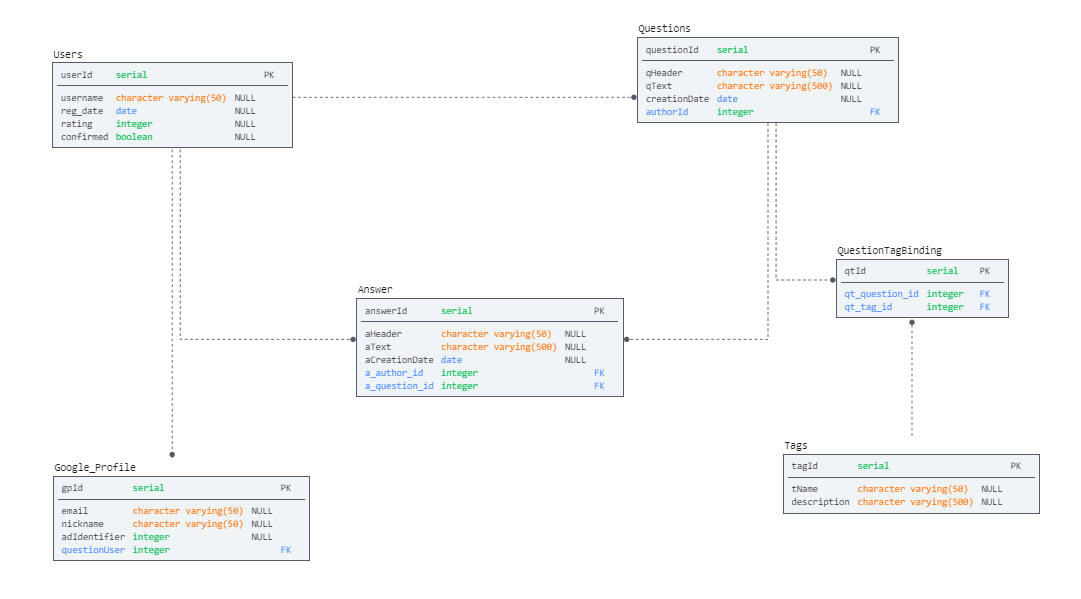
1. Реалізувати функції внесення, редагування та вилучення даних у таблицях бази даних, створених у лабораторній роботі №1, засобами консольного інтерфейсу.
2. Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі.
3. Забезпечити реалізацію пошуку за декількома атрибутами з двох та більше сутностей одночасно: для числових атрибутів – у рамках діапазону, для рядкових – як шаблон функції LIKE оператора SELECT SQL, для логічного типу – значення True/Fаlse, 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 відповідність 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 відповідності 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 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**!
3. Для реалізації пошуку необхідно підготувати 3 запити, включають дані з декількох таблиць і фільтрують рядки за 3-4 атрибутами цих таблиць. Забезпечити можливість введення 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). При цьому модель, подання та контролер мають бути реалізовані у окремих файлах. Для доступу до бази даних використовувати **лише мову SQL** (без ORM).

**Логічна модель бази даних**

Нижче (Рис. 1) наведено логічну модель бази даних:



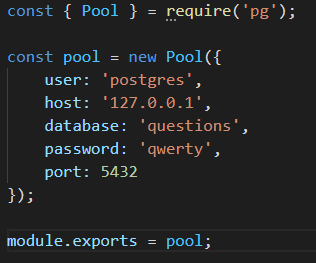
Зміни у порівнянні з першою лабораторною роботою:

1. В таблицю Users додано атрибут **confirmed** типу **boolean.**
2. Для зручності, було перейменовано назви атрибутів у таблицях.

**Середовище розробки та налаштування підключення до бази даних**

Для виконання лабораторної роботи використовувалась мова програмування TypeScript та текстовий редактор Visual Studio Code.

Для підключення до серверу бази даних PostgreSQL використовувався **npm** **пакет** **pg**. Для цього він підключався до проекту, з нього було «дістано» класс **Pool** на його основі було створено змінну для керування базою даних:



**Опис структури програми**

Програма складається із: 8 моделей, що містять класи для взаємодії з базою даних, 3 допоміжних модуля в папці **utils**, модуль **view** для відображення меню, модуль **db**, що забезпечує підключення до бази даних, модуль **types** із оголошеними типами для зручності роботи з таблицями, **controller**, який містить в собі меню та забезпечує взаємодію моделей та представлення, модуль **main**, який запускає метод модуля **controller** та розпочинає роботу програми.

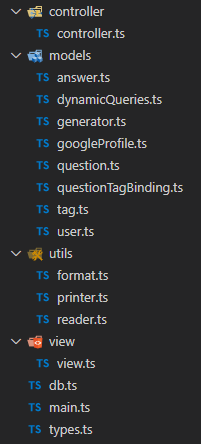
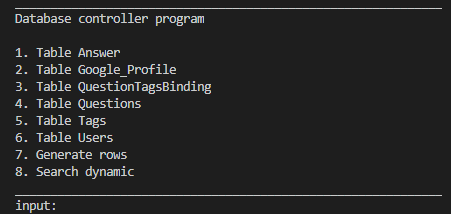


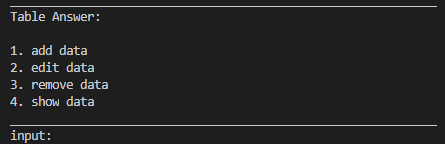
Рис. 2 – Структура програмного коду

**Структура меню програми**

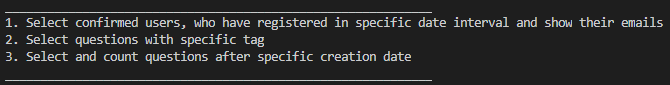
Головне меню



Меню для таблиці



Меню для вибору динамічних запитів



Меню вибору кількості рядків для генерації



Посилання для навігації

1. [Лістинги програми з директивами внесення, редагування та вилучення даних у базі даних та результати виконання цих директив.](#лістингивнесенняідт)

a. [Функції внесення](#внесення)

b. [Функції редагування](#редагування)

c. [Функції видалення](#видалення)

2. [Лістинги програми з директивами рандомізованого внесення даних до таблиці Users та результати.](#рандом)

3. [Лістинги програми з директивами динамічних запитів та валідації даних.](#динамічні)

4. [Обробка виняткових ситуацій(помилок) при введенні/вилученні та валідації даних.](#динамічні)

5. [Дослідження режимів ON DELETE.](#дослідження)

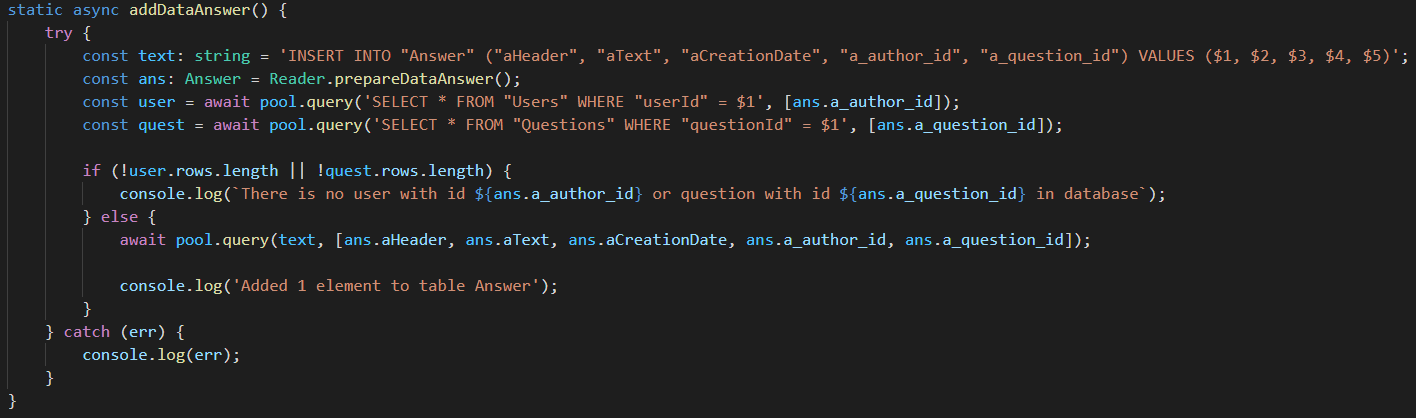
6. [Ілюстрації програмного коду на Github.](#Ілюстрації)

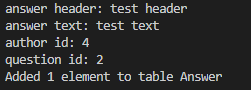
**Лістинги програми з директивами внесення, редагування та вилучення даних у базі даних та результати виконання цих директив**

Для кожної таблиці було створено клас-модель, що містить методи для роботи з відповідною таблицею бази даних.

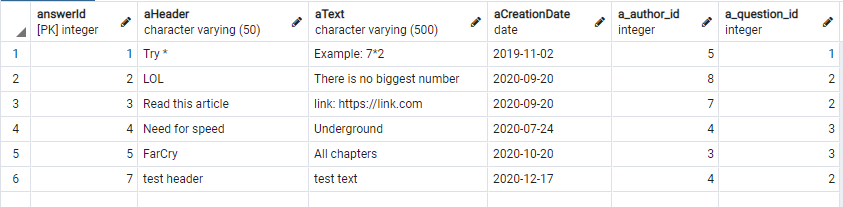
**Функції внесення**

Додавання запису у таблицю Answer:



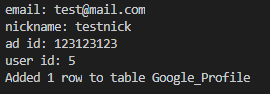


Результат:



Додавання запису у таблицю Google\_Profile:

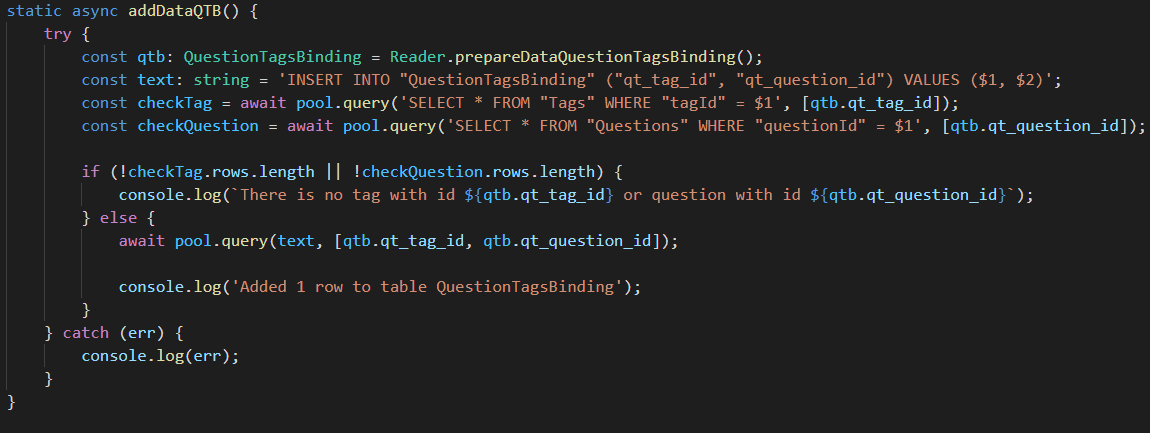


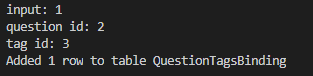


Результат:

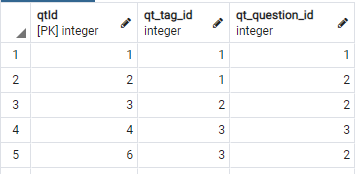


Додавання запису у таблицю QuestionTagsBinding:

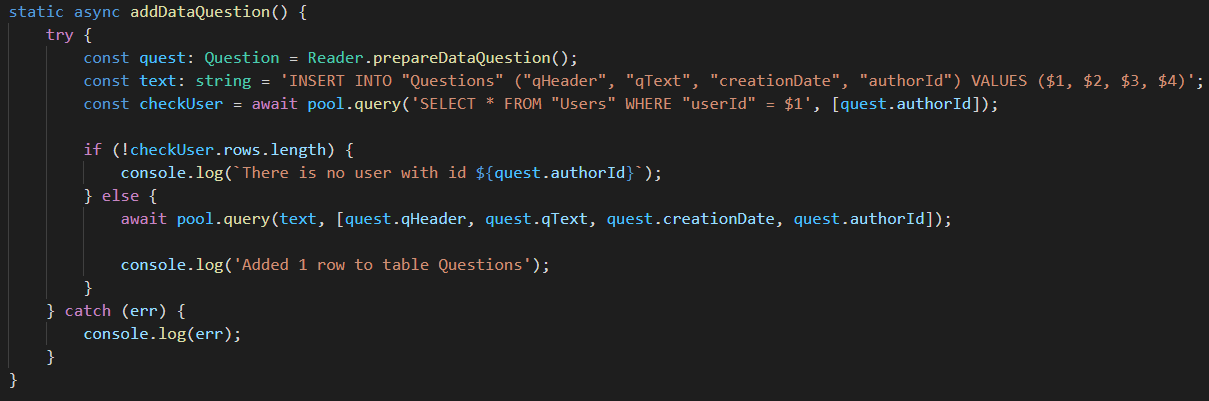


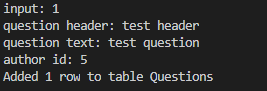


Результат:

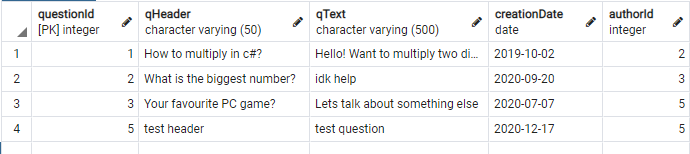


Додавання запису у таблицю Questions:

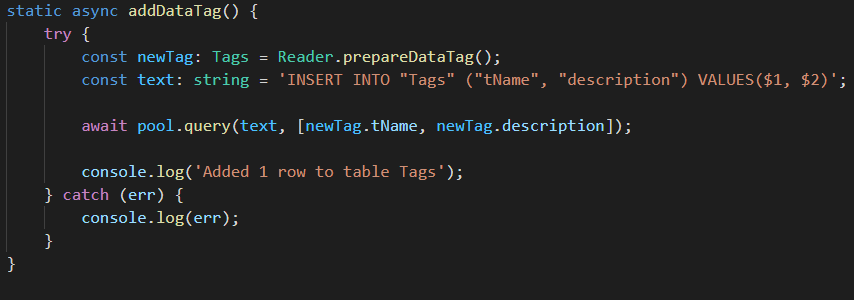


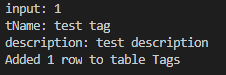


Результат:

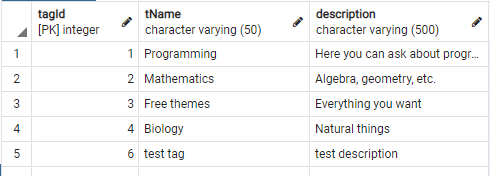


Додавання запису у таблицю Tags:

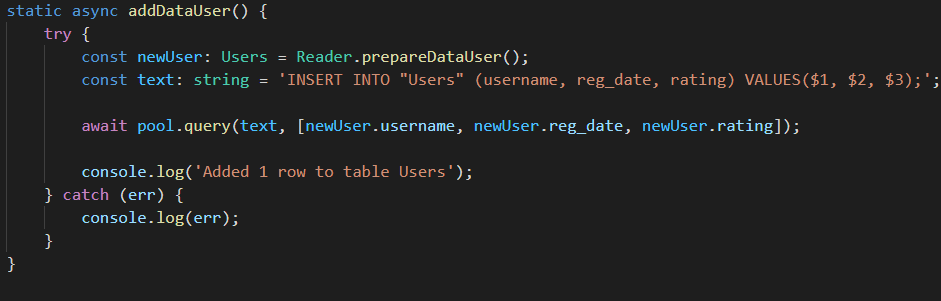


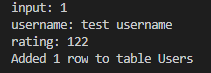


Результат:

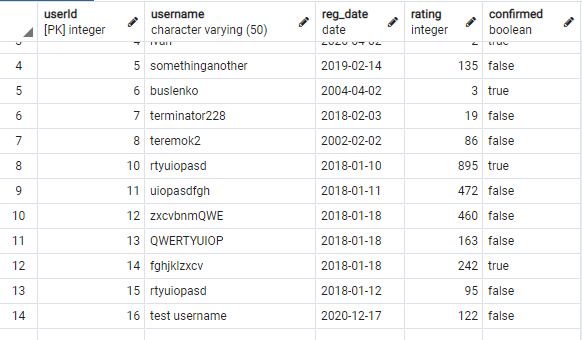


Додавання запису у таблицю Users:



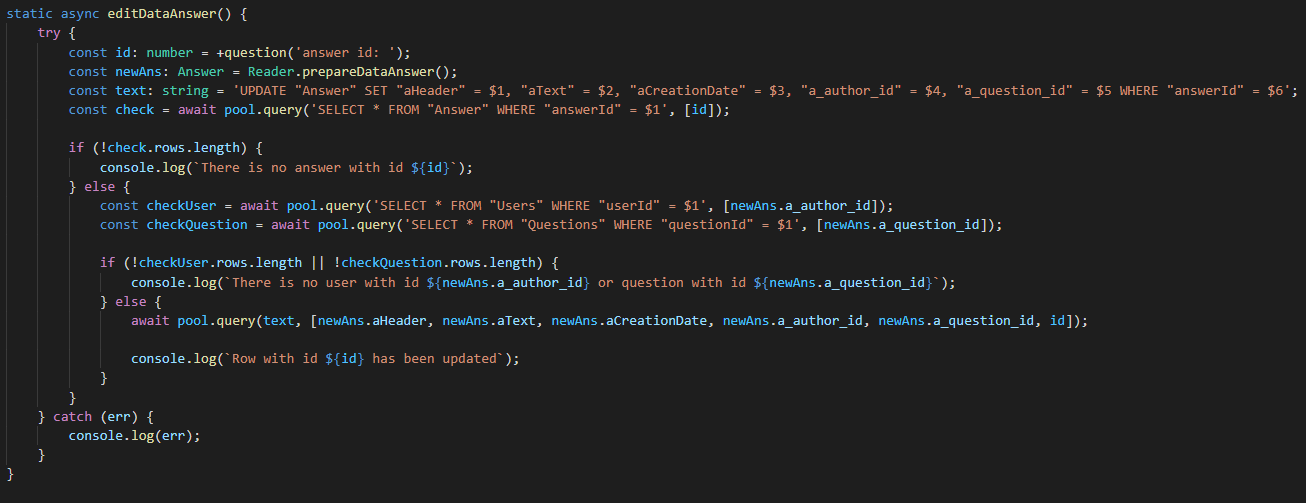


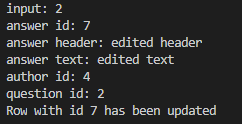
Результат:



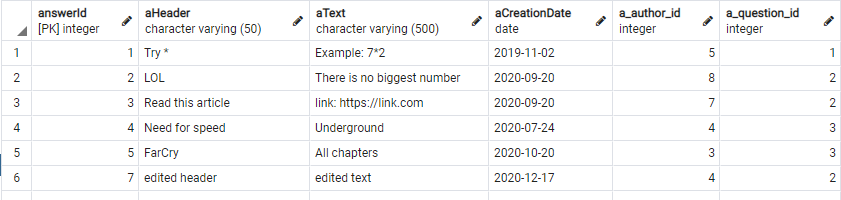
**Функції редагування**

Редагування запису в таблиці Answer:

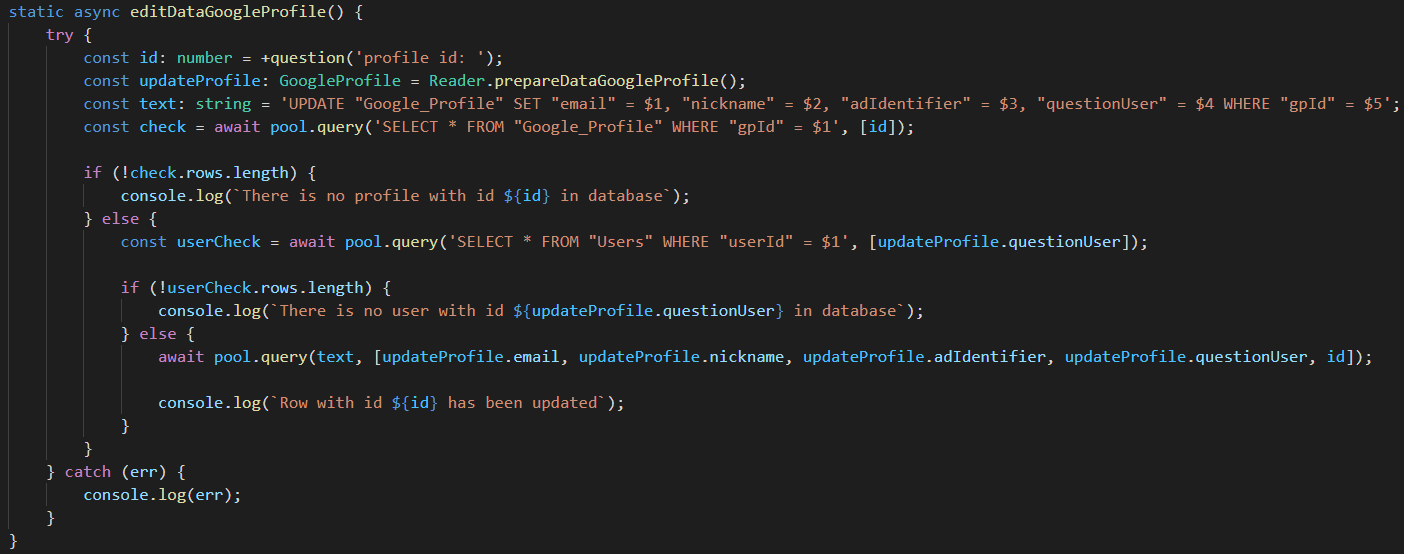


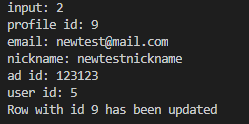


Результат:



Редагування запису в таблиці Google\_Profile:

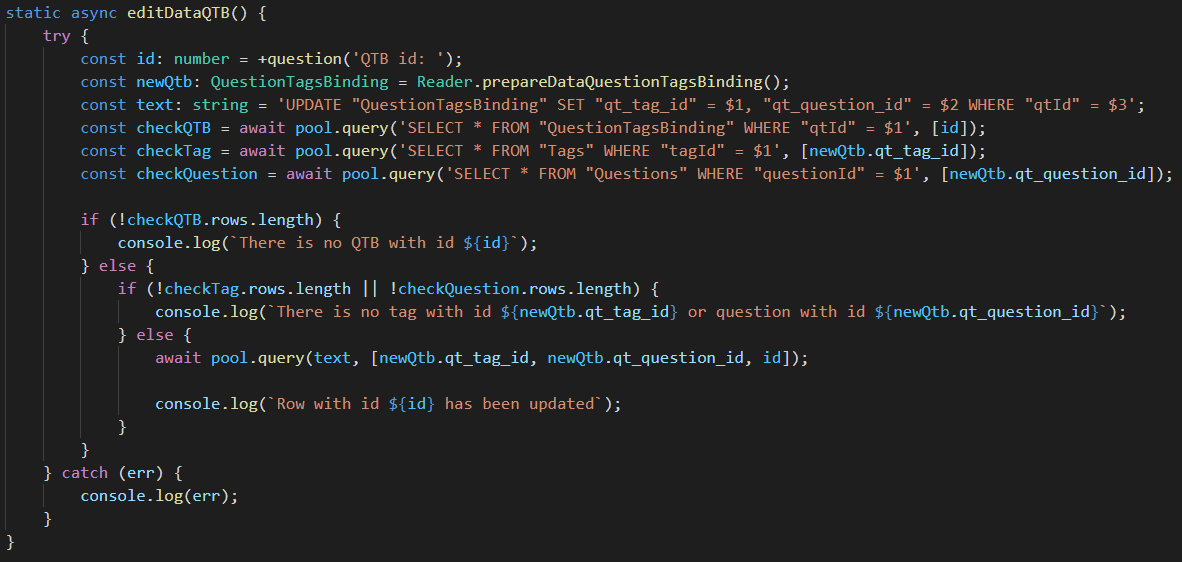


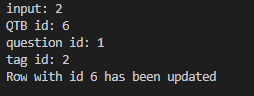


Результат:

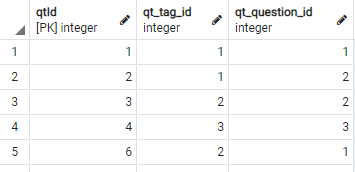


Редагування запису в таблиці QuestionTagsBinding:

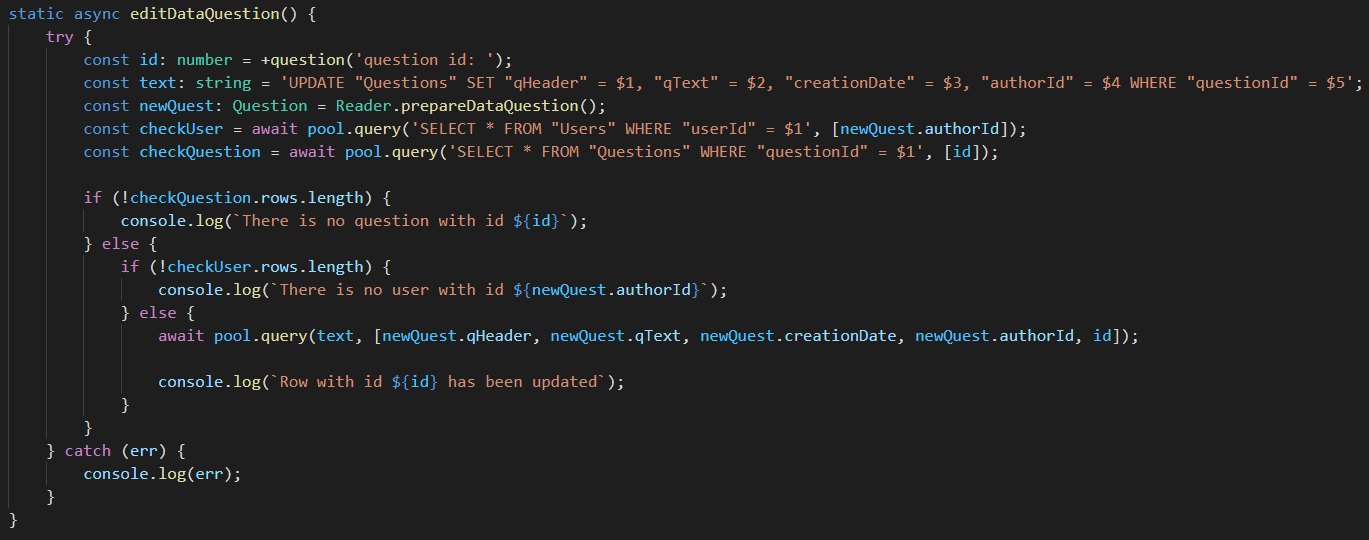


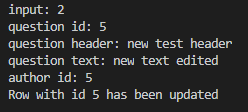


Результат:

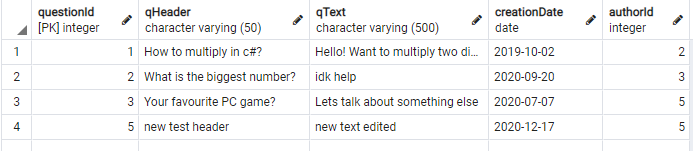


Редагування запису в таблиці Questions:

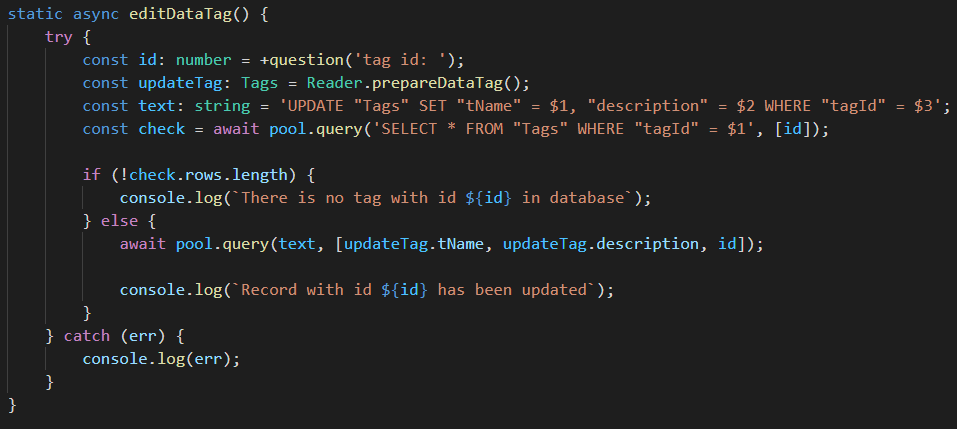


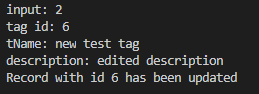


Результат:

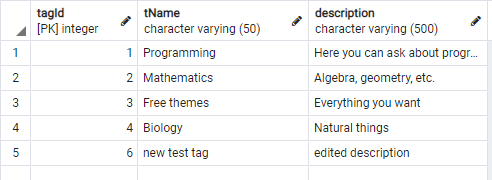


Редагування запису в таблиці Tags:

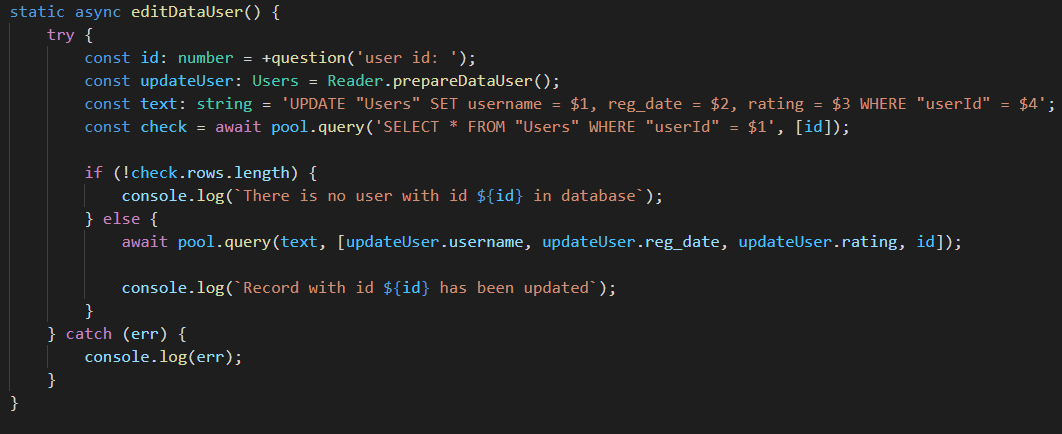


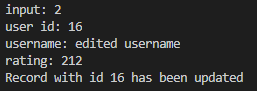


Результат:

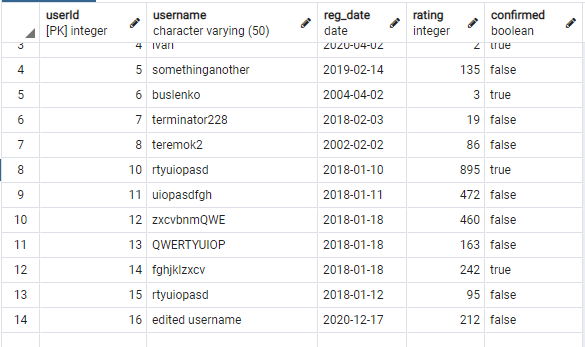


Редагування запису в таблиці Users:



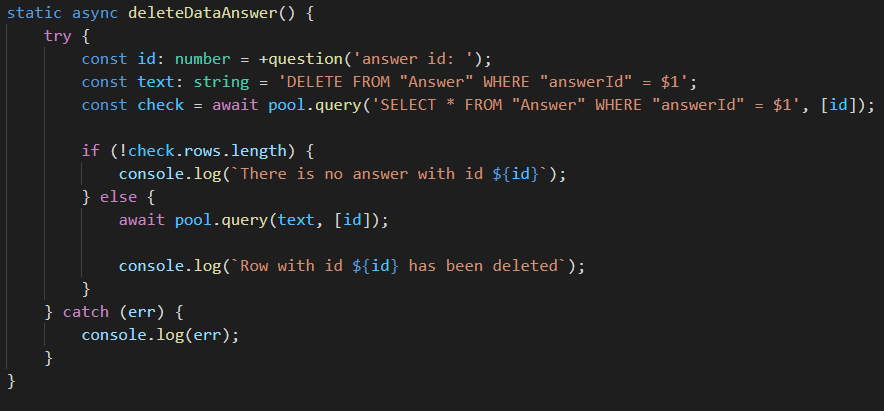


Результат:



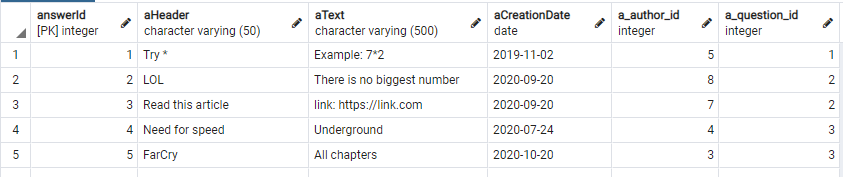
**Функції видалення**

Видалення в таблиці Answer:

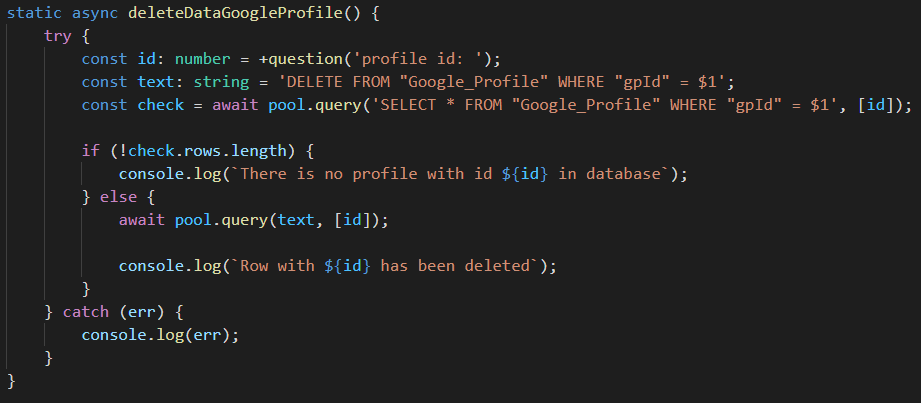




Результат:



Видалення в таблиці Google\_Profile:

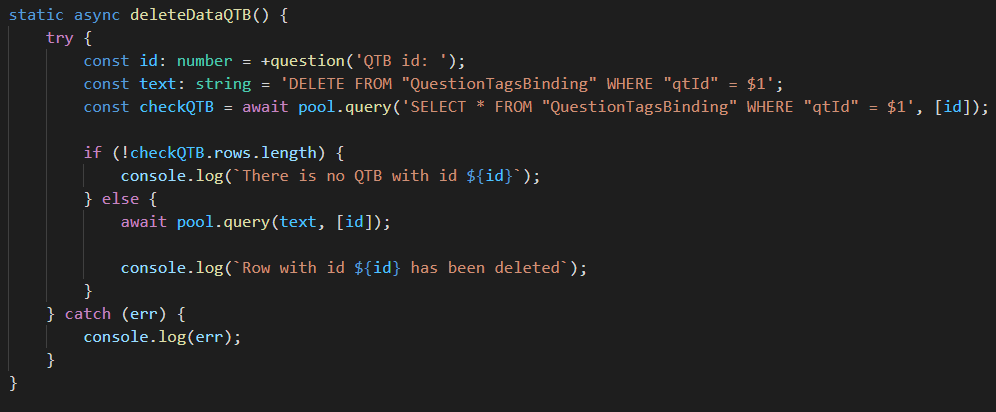




Результат:

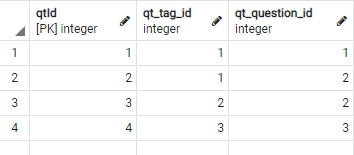


Видалення в таблиці QuestionTagsBinding:

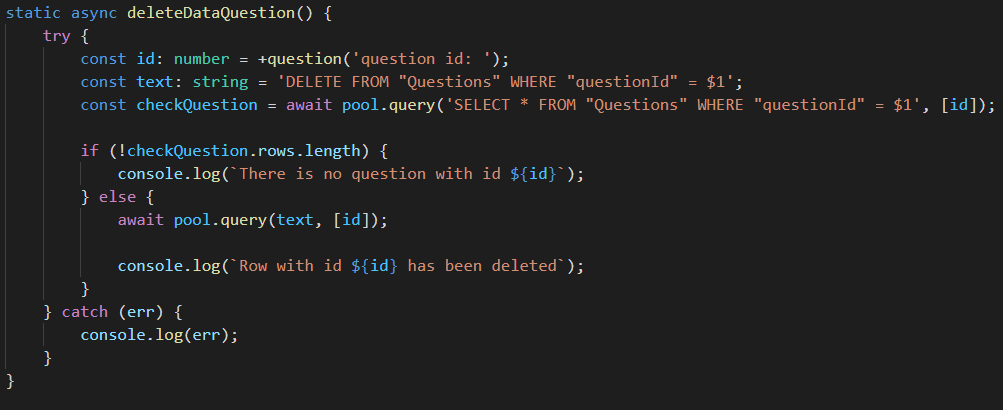




Результат:

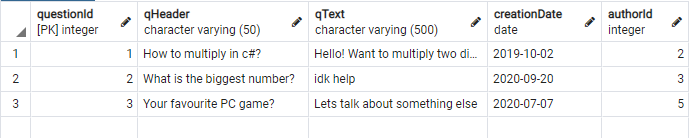


Видалення в таблиці Questions:

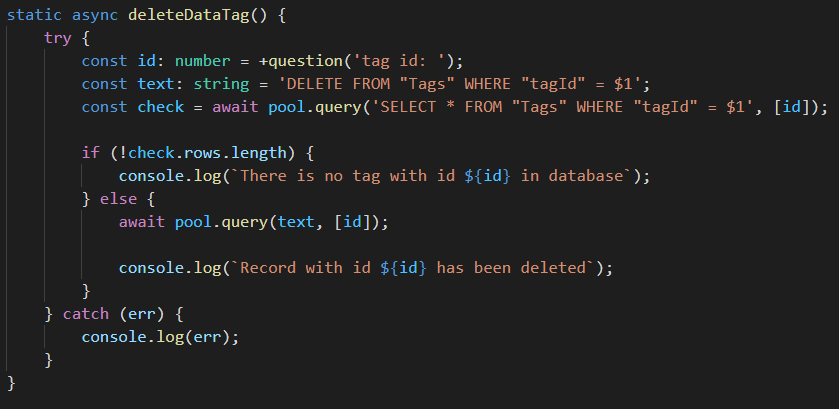




Результат:

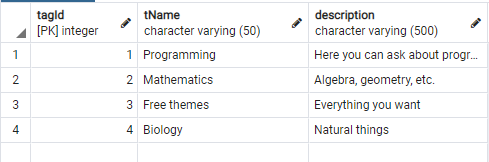


Видалення в таблиці Tags:

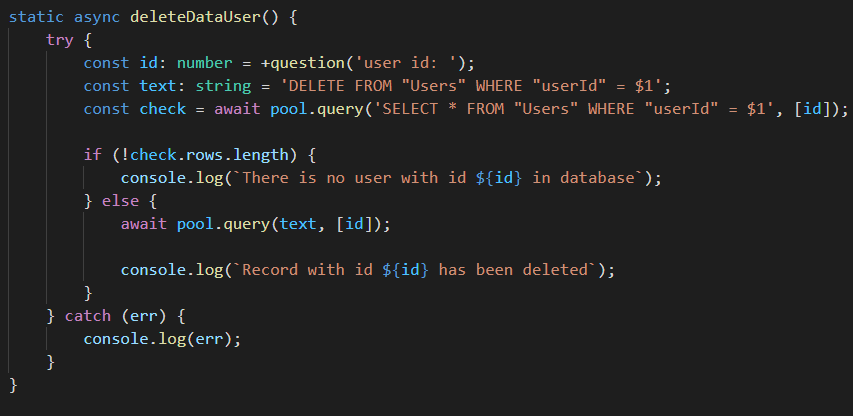




Результат:

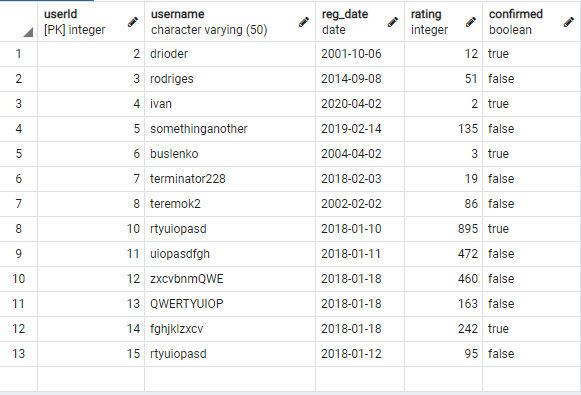


Видалення в таблиці Users:





Результат:



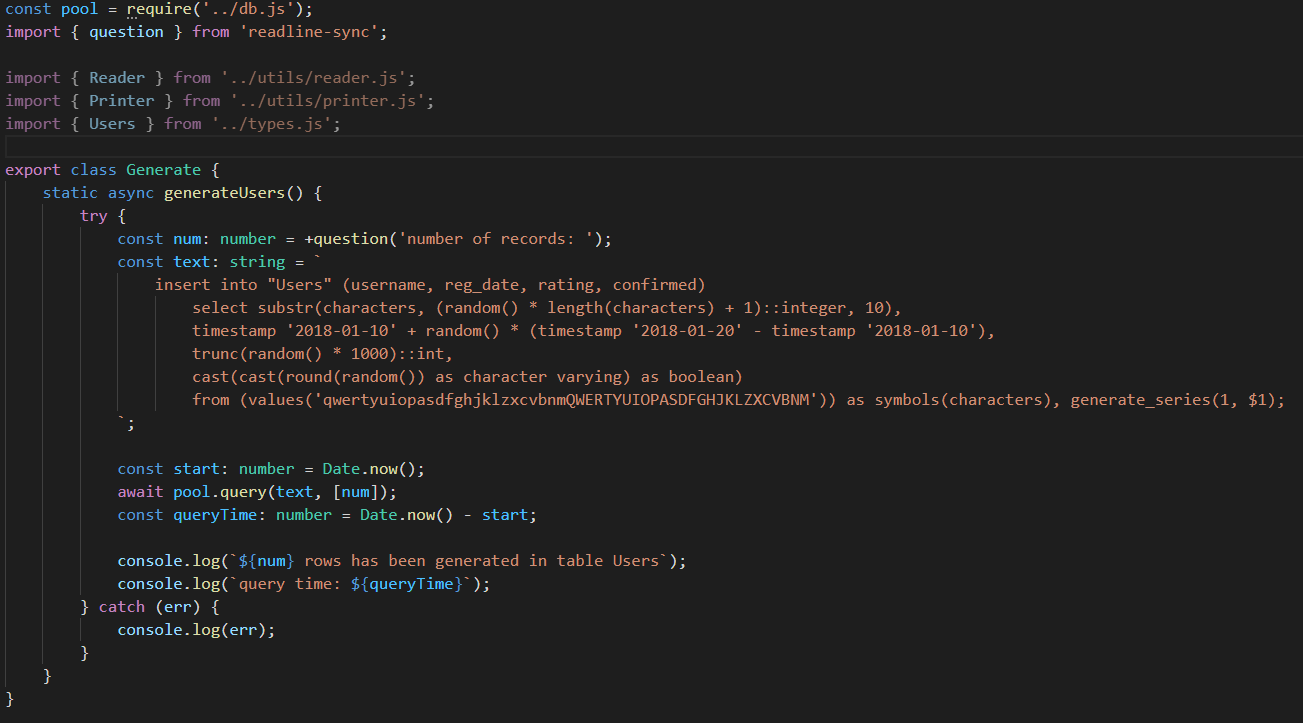
**Лістинги програми з директивами внесення рандомізованих даних**

**і виконання динамічних запитів у базі даних та результати виконання**

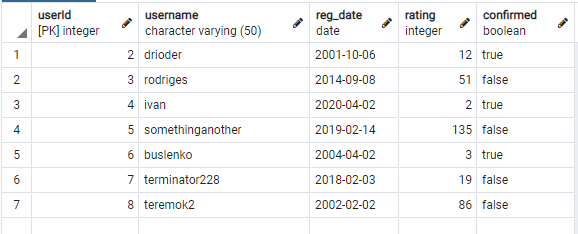
**цих директив**

**Рандомізоване внесення даних до таблиці Users**

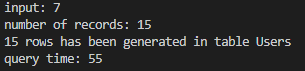
Було створено клас Generate із методом generateUsers. Задана кількість записів генерується у таблиці **Users**:



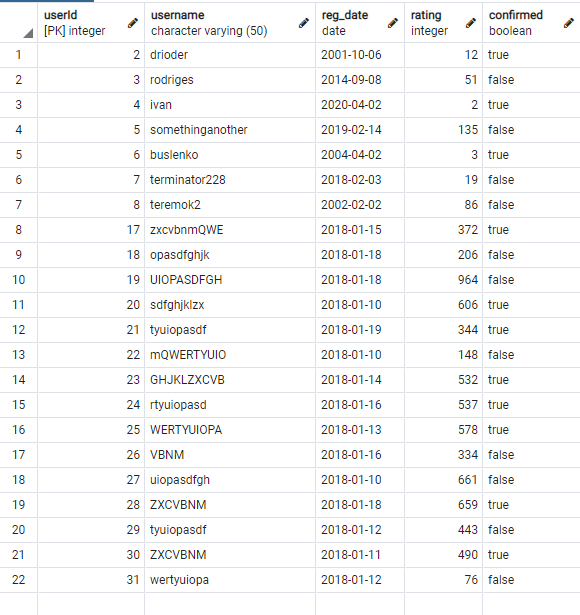
Таблиця Users до генерації:



Генеруємо записи:

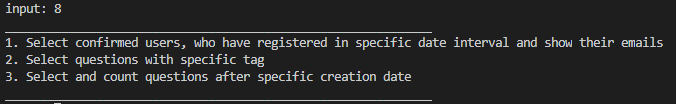


Таблиця Users після генерації:



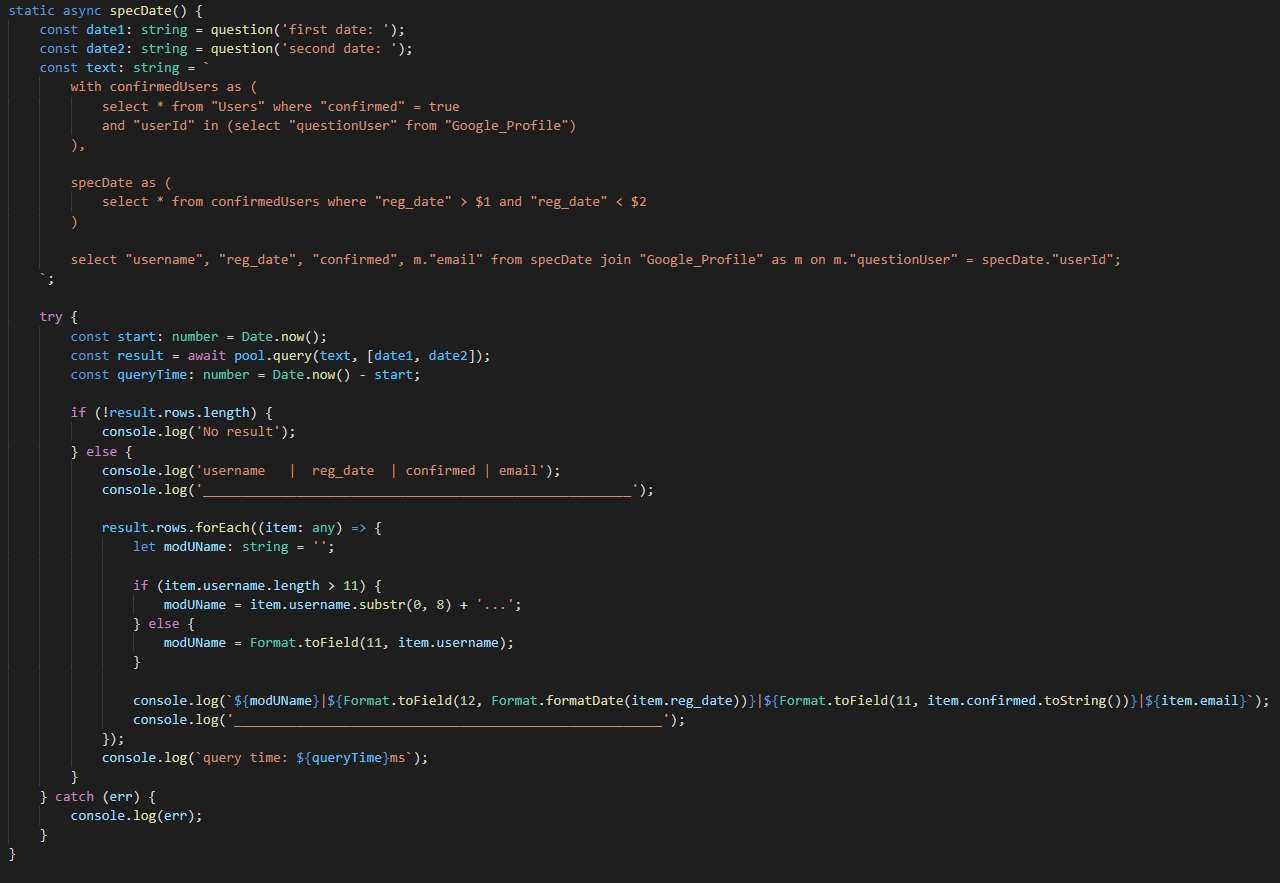
**Виконання динамічних запитів бази даних**

Меню вибору запиту:

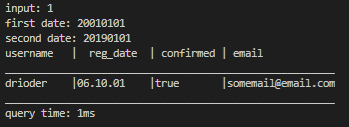


Обрахунок часу запиту відбувається безпосередньо у кожному методі моделі, що обробляє запит. Береться поточний час ДО відсилання запиту до бази даних і виводиться різниця між поточним часом ПІСЛЯ запиту і часом ДО запиту, щоб не враховувати час на форматування виводу у консоль.

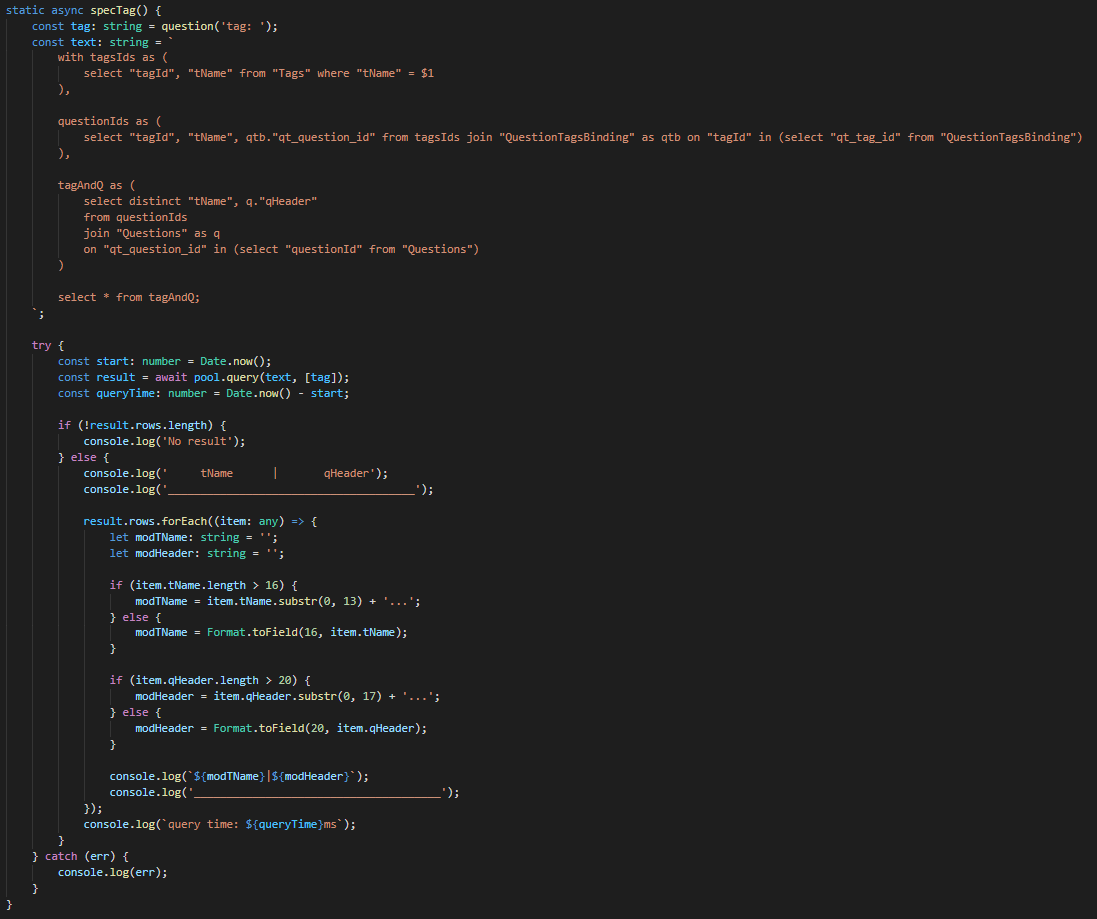
Перший запит:



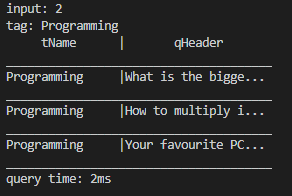
Результат:



Другий запит:



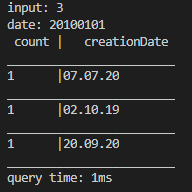
Результат:



Третій запит:



Результат:

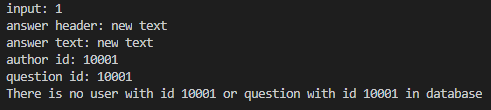


**Обробка виняткових ситуацій (помилок) при введенні/вилученні**

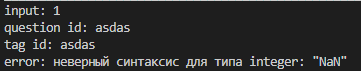
**та валідації даних**

Обробка виняткових ситуацій при введенні та видаленні даних виконується за допомогою блоків try-catch та перевірочних запитів перед основним запитом.

Додавання рядка із неіснуючим зовнішнім ключем:



Введення рядка з полем, тип якого не відповідає дійсному:



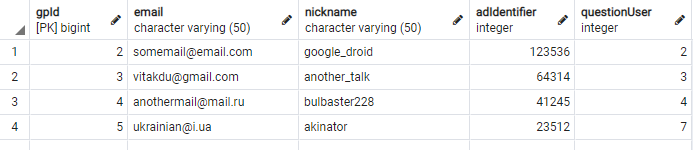
Видалення рядка, ключ якого є зовнішнім ключем іншої таблиці:



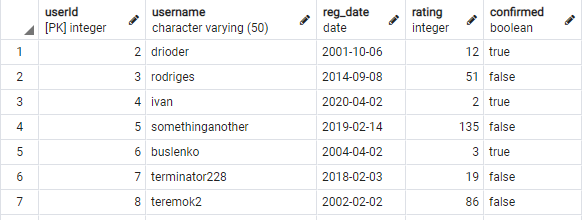
**Дослідження режимів обмеження ON DELETE**

Дослідження режимів будемо проводити на таблиці Users (батьківська) та Google\_Profile (дочірня). Використаємо запис із таблиці Google\_Profile з gpId = 5 та зовнішнім ключем questionUser = 7:

Таблиця Google\_Profile:



Таблиця Users:



Режим NO ACTION

При видаленні запису із таблиці Users, id якого присутній в таблиці Google\_Profile отримуємо повідомлення про помилку:



Режим SET NULL

При видаленні запису із таблиці Users, id якого присутній в таблиці Google\_Profile, якщо на questionUser таблиці Google\_Profile NOT NULL, отримуємо повідомлення:



Якщо прибрати NOT NULL, то questionUser прийме значення NULL.

Режим SET DEFAULT



В налаштуваннях Users поле DEFAULT не заповнено, SET DEFAULT намагається встановити його як null, проте це порушує обмеження поля NOT NULL.

Режим RESTRICT

При видаленні запису із таблиці Users, id якого присутній в таблиці Google\_Profile отримуємо таке саме повідомлення про помилку, як і в режимі NO ACTION:

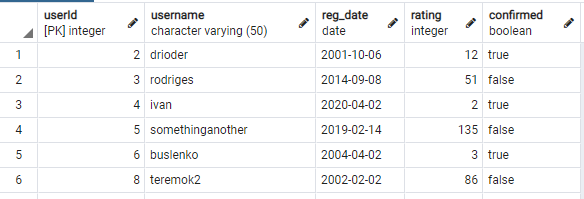


Режим CASCADE

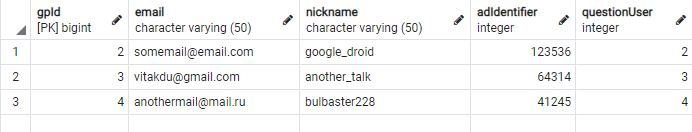
При видаленні запису із таблиці Users, id якого присутній в таблиці Google\_Profile, видаляється запис таблиці Google\_Profile та запис таблиці Users із відповідним userId:



Таблиця Users:



Таблиця Google\_Profile:



**Ілюстрації програмного коду на Github**