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-82

Шеремет Д. О.

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:

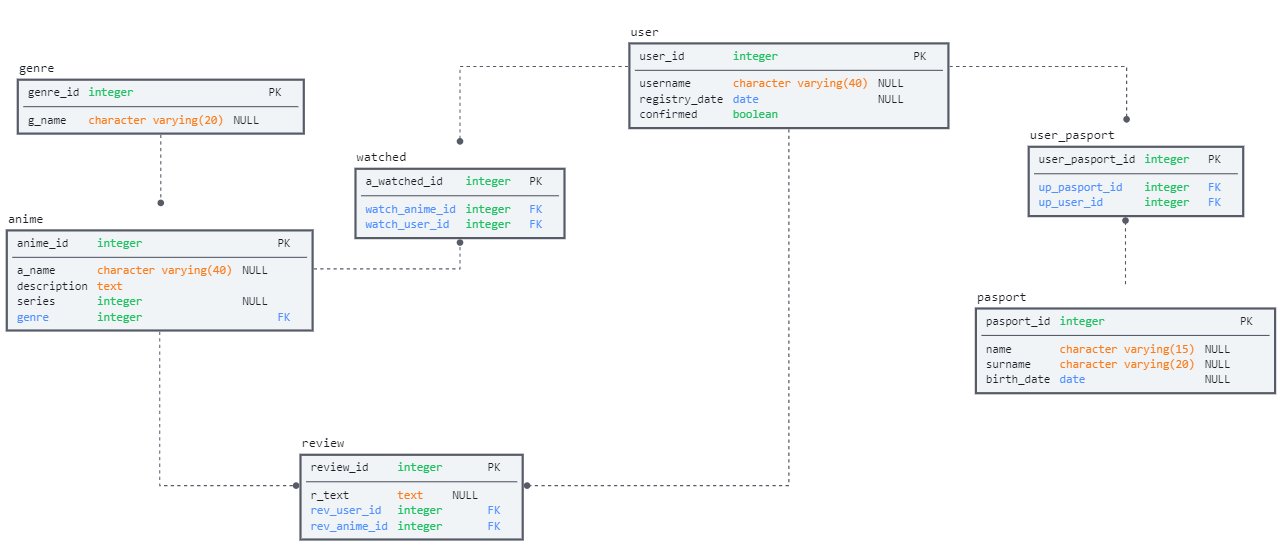
****

Рис. 1 – Логічна модель бази даних

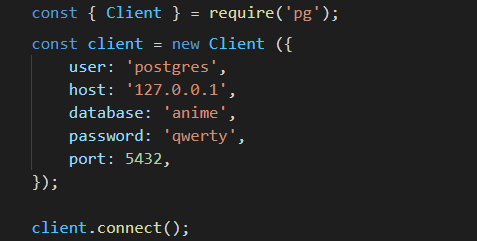
Зміни у порівнянні з першою лабораторною роботою:

1. В таблицю user додано атрибут **confirmed** типу **boolean.**
2. Назви атрибутів зовнішніх ключів **user\_id** та **anime\_id** таблиці **review** булоперейменованоу **rev\_user\_id** та **rev\_anime\_id** відповідно.
3. Назви атрибутів зовнішніх ключів **user\_id** та **passport\_id** таблиці **user\_passport** булоперейменованоу **up\_user\_id** та **up\_passport\_id** відповідно.

**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 **Client** на його основі було створено змінну для керування базою даних:



**Опис структури програми**

Програма містить 4 основних модулі: **model.ts, view.ts, controller.ts –** дані, інтерфейс та обробка даних відповідно, **main.ts –** запуск програми, та один допоміжний модуль **types.ts** в якому описані типи.

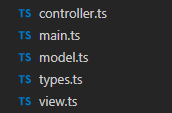
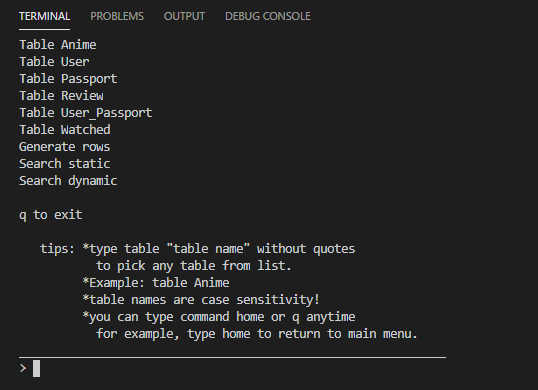


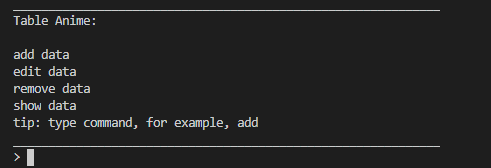
Рис. 2 – Структура програмного коду

**Структура меню програми**

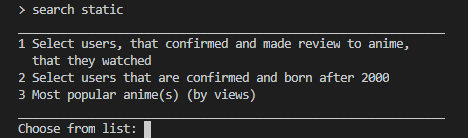
Головне меню



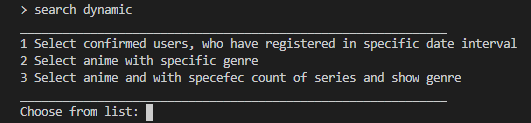
Меню для таблиці



Меню для вибору статичних запитів



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.](#Лістинги)
   1. [Функції внесення](#Внесення)
   2. [Функції редагування](#Редагування)
   3. [Функції видалення](#Видалення)
2. [Лістинги програми з директивами рандомізованого внесення даних до таблиці user та результати.](#Рандом)
3. [Лістинги програми з директивами динамічних запитів та валідації даних.](#Динамічний)
4. [Обробка виняткових ситуацій(помилок) при введенні/вилученні та валідації даних.](#Помилки)
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**

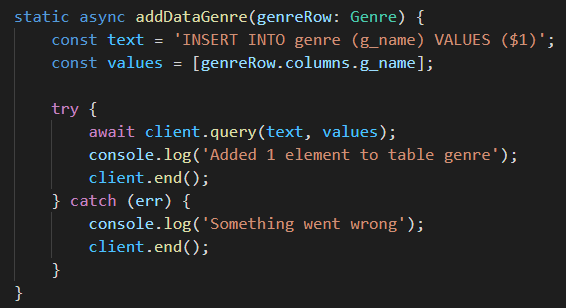
**Функції внесення**

Додавання відбувається через метод **add,** що має доступ до змінної, в якій зберігається назва обраної таблиці, за допомогою якої обираються методи підготовки та додавання даних для конкретної таблиці:



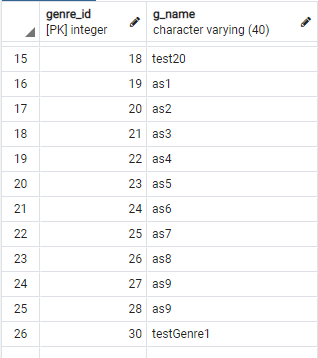
Рис. 3 – Лістинг методу вибору таблиці для додавання елемента – **add**

Додавання запису в таблицю genre

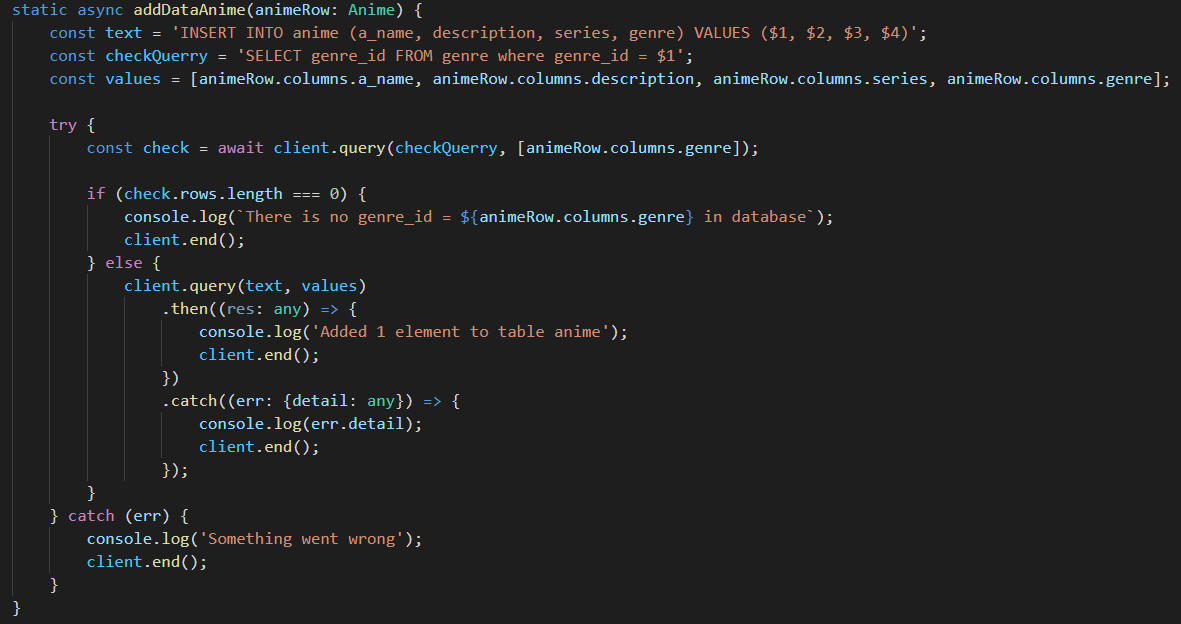


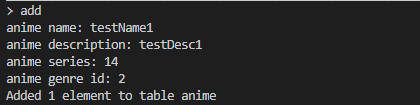


Результат

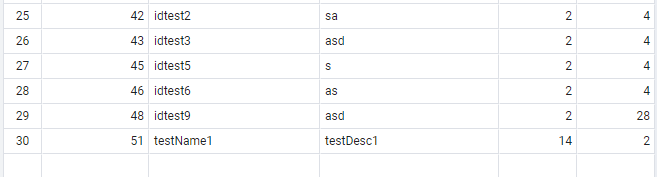


Додавання запису в таблицю anime

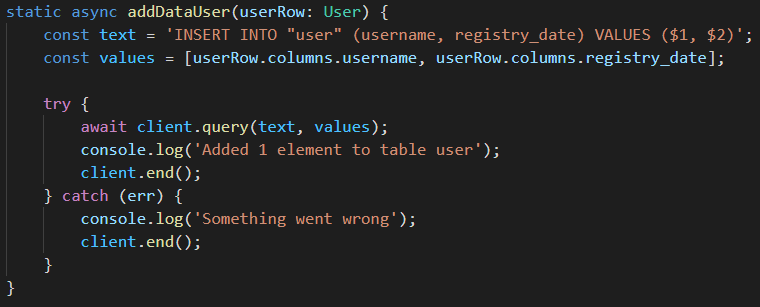




Результат:

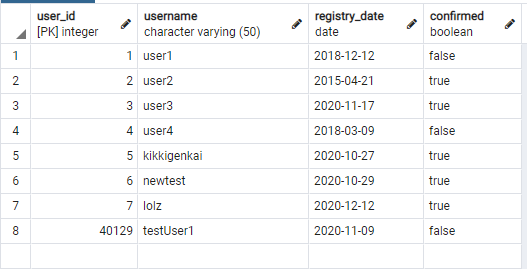


Додавання запису в таблицю user

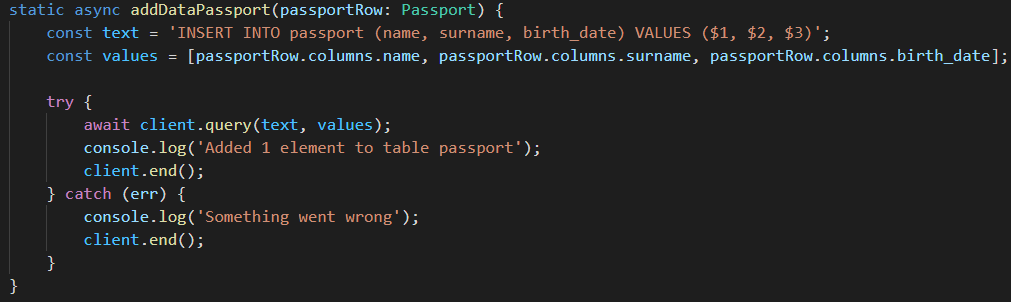


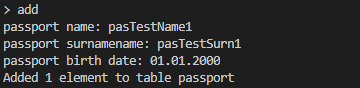


Результат:

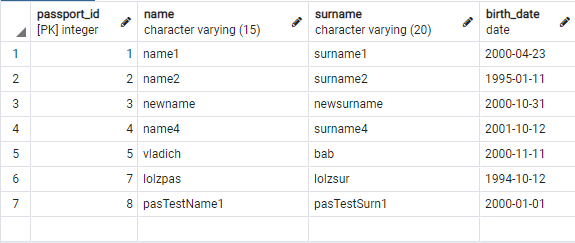


Додавання запису в таблицю passport

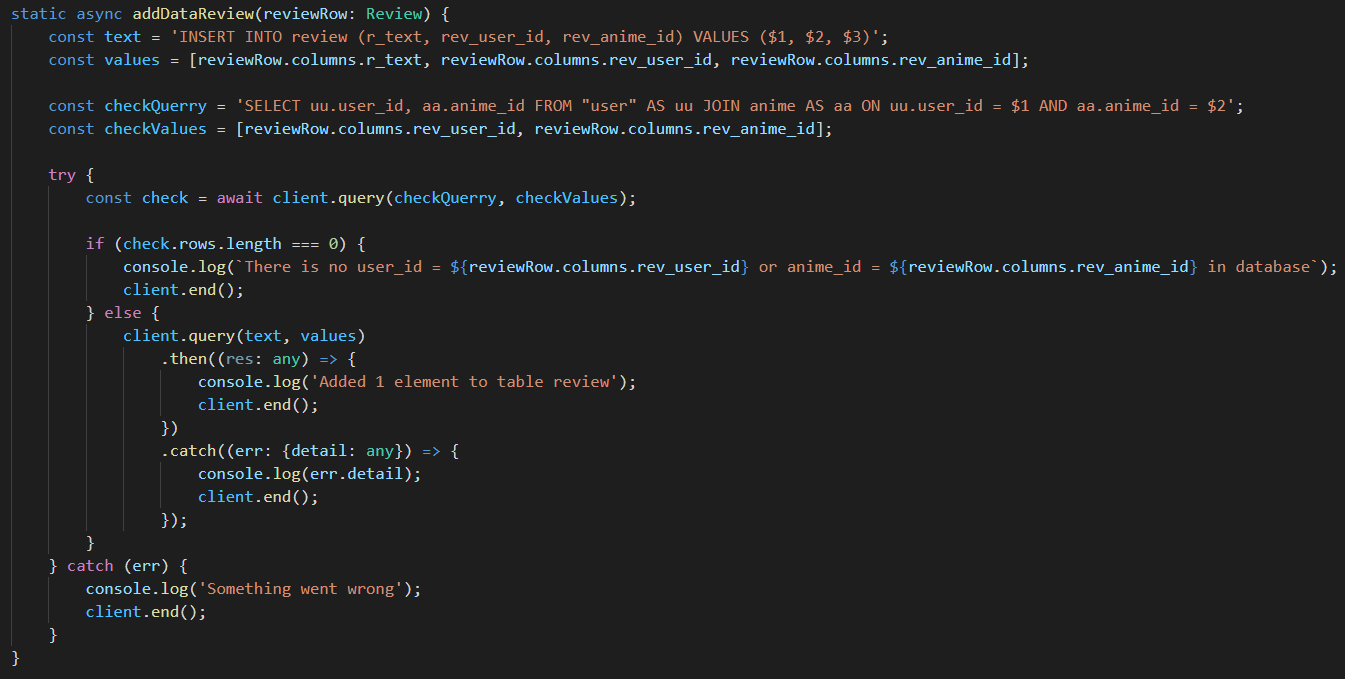


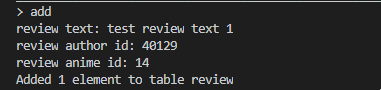


Результат:

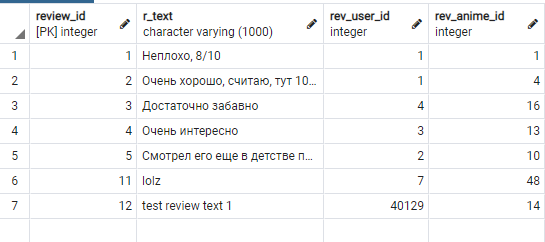


Додавання запису в таблицю review

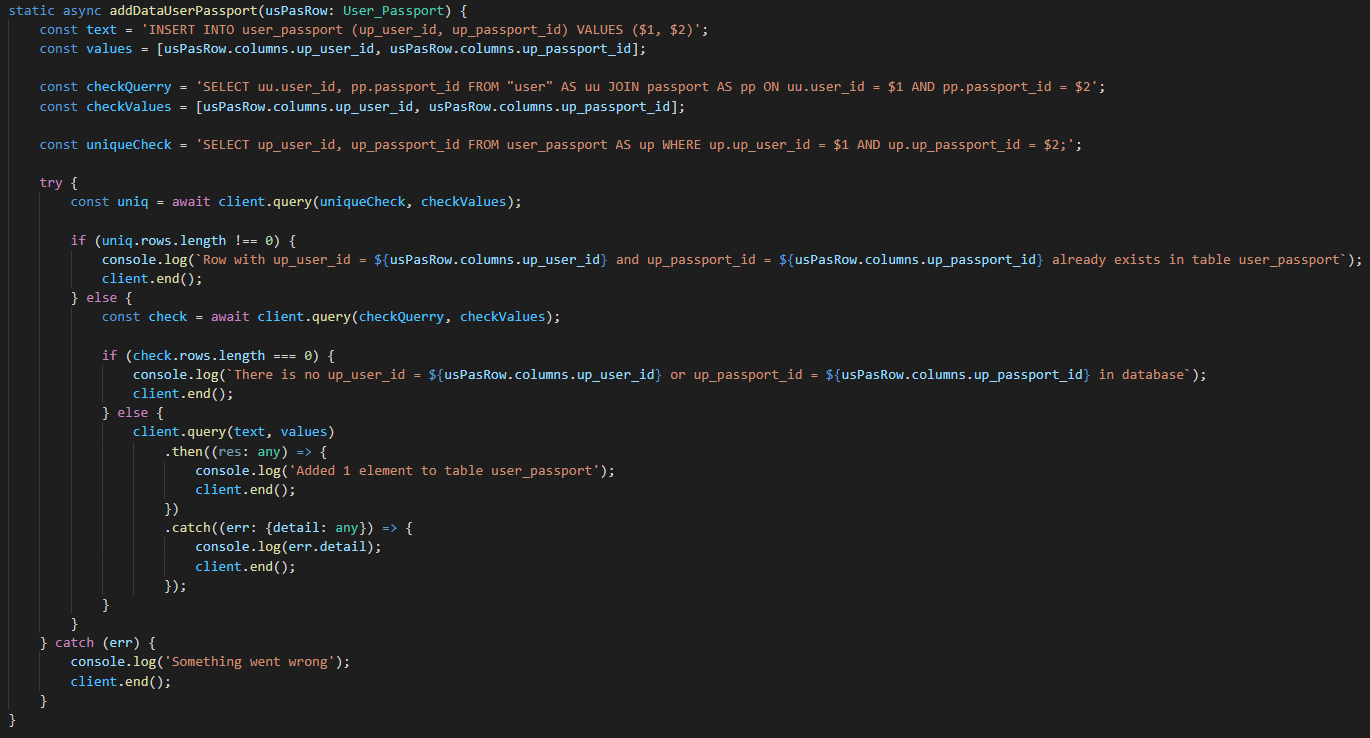


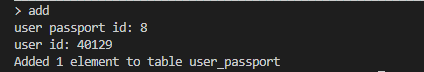


Результат:

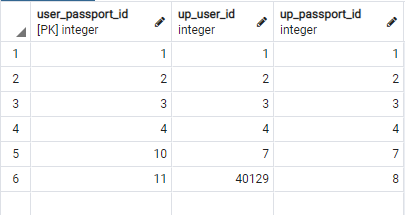


Додавання запису в таблицю user\_passport

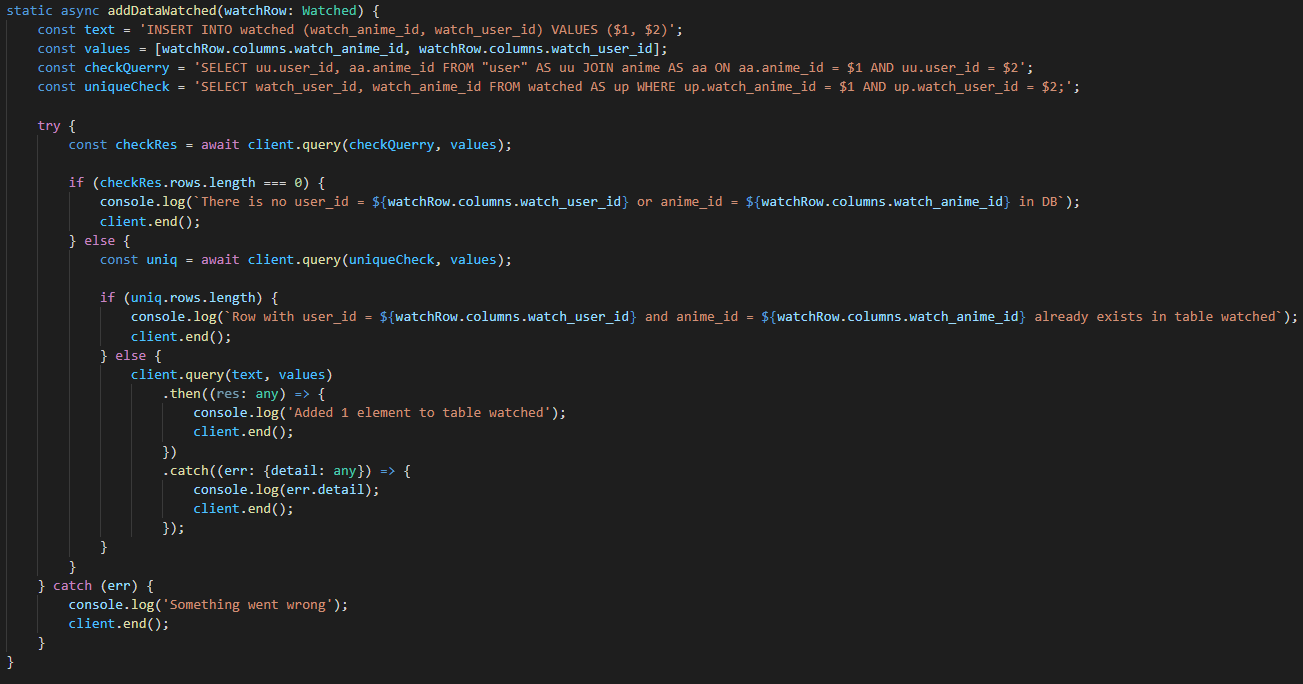


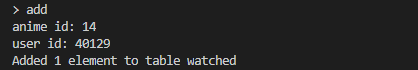


Результат:

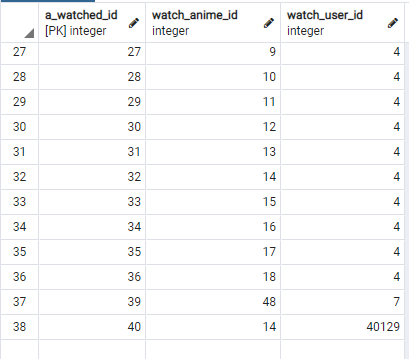


Додавання запису в таблицю watched



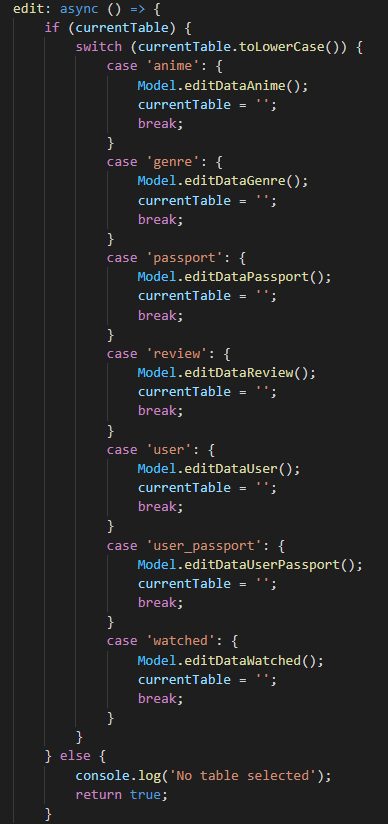


Результат:

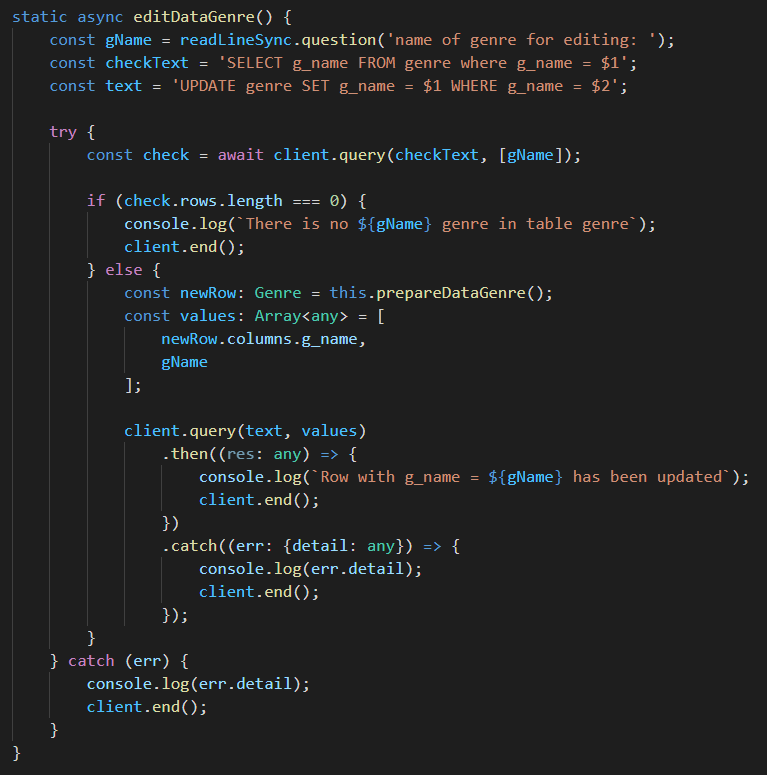


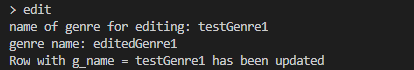
**Функції для редагування**

Редагування відбувається через метод **edit,** що має доступ до змінної, в якій зберігається назва обраної таблиці, за допомогою якої обирається метод редагування даних для конкретної таблиці:

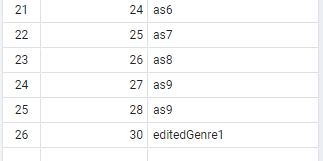


Редагування запису в таблиці genre

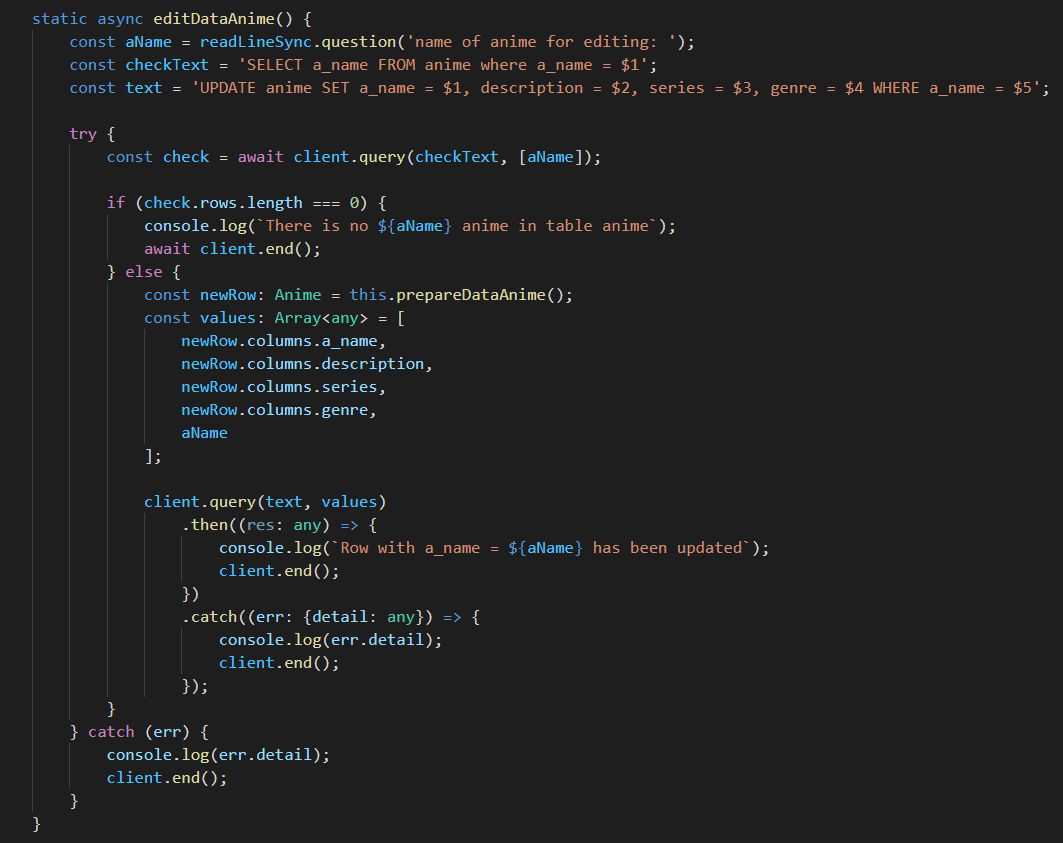


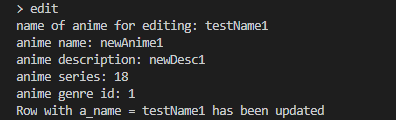


Результат:

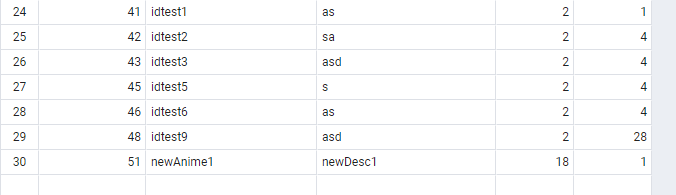


Редагування запису в таблиці anime

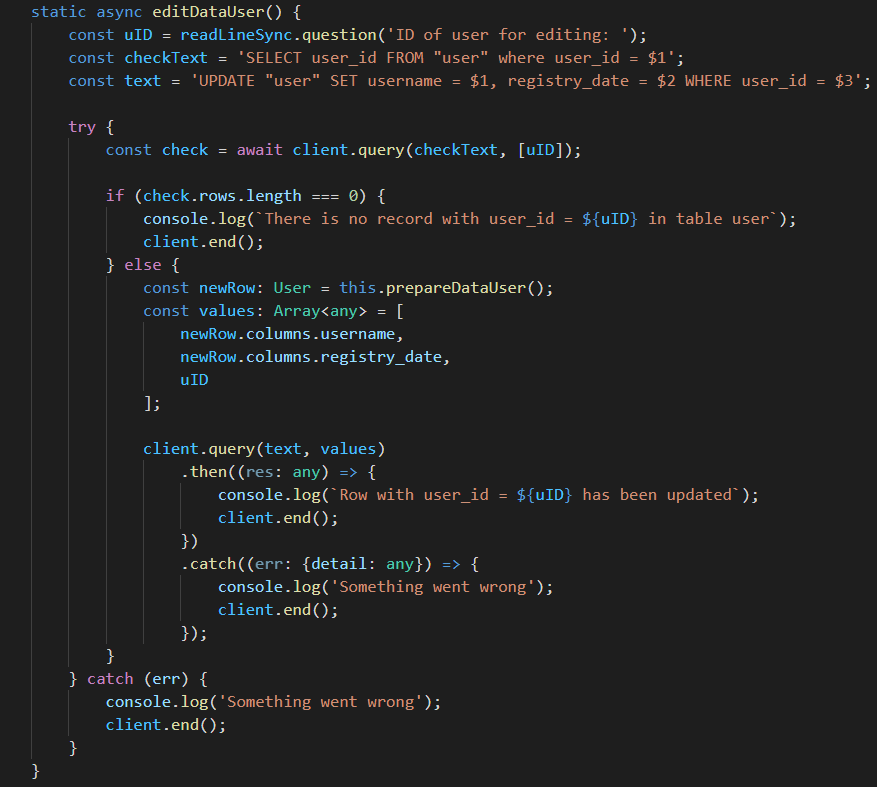


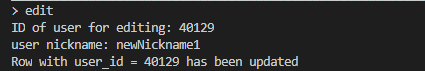


Результат:

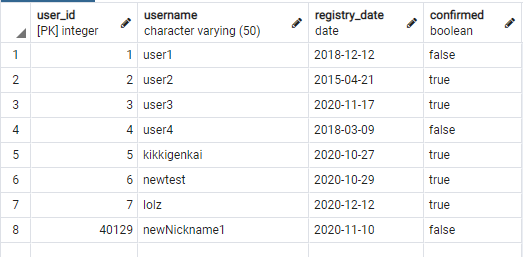


Редагування запису в таблиці user

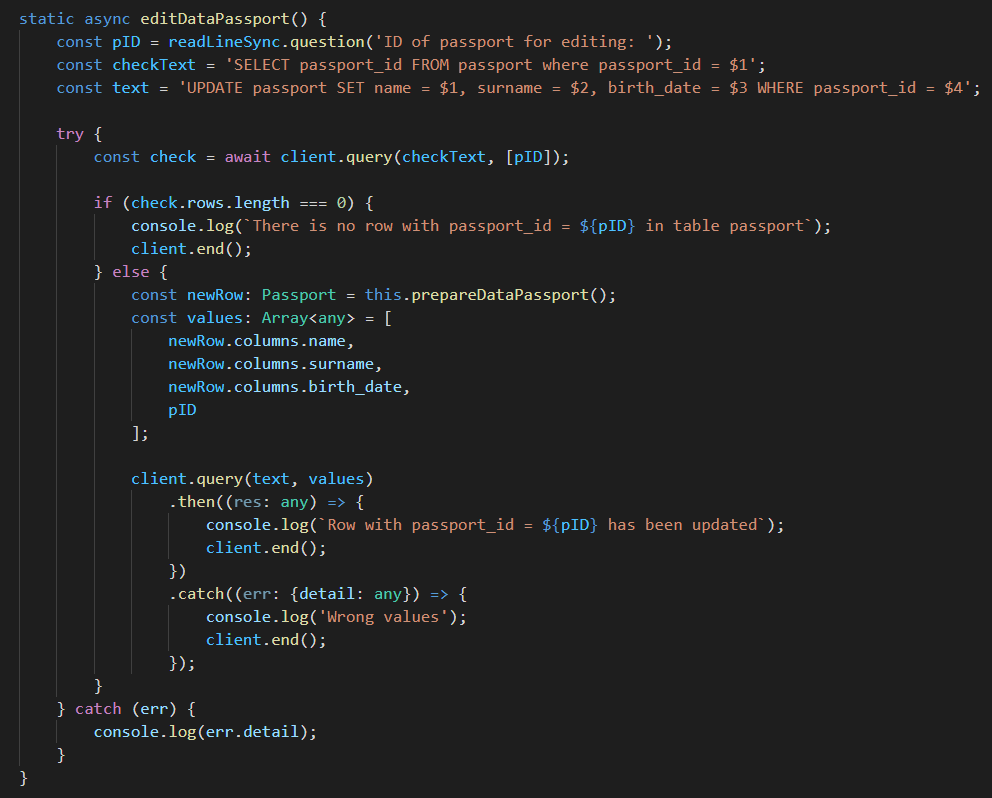


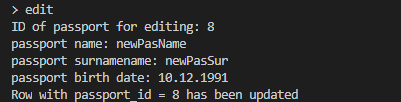


Результат:

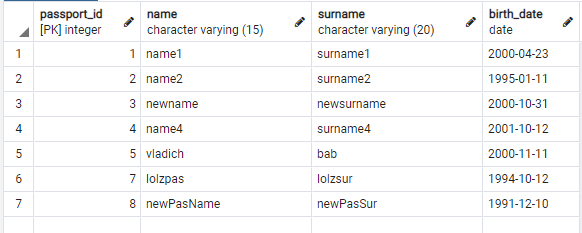


Редагування запису в таблиці passport

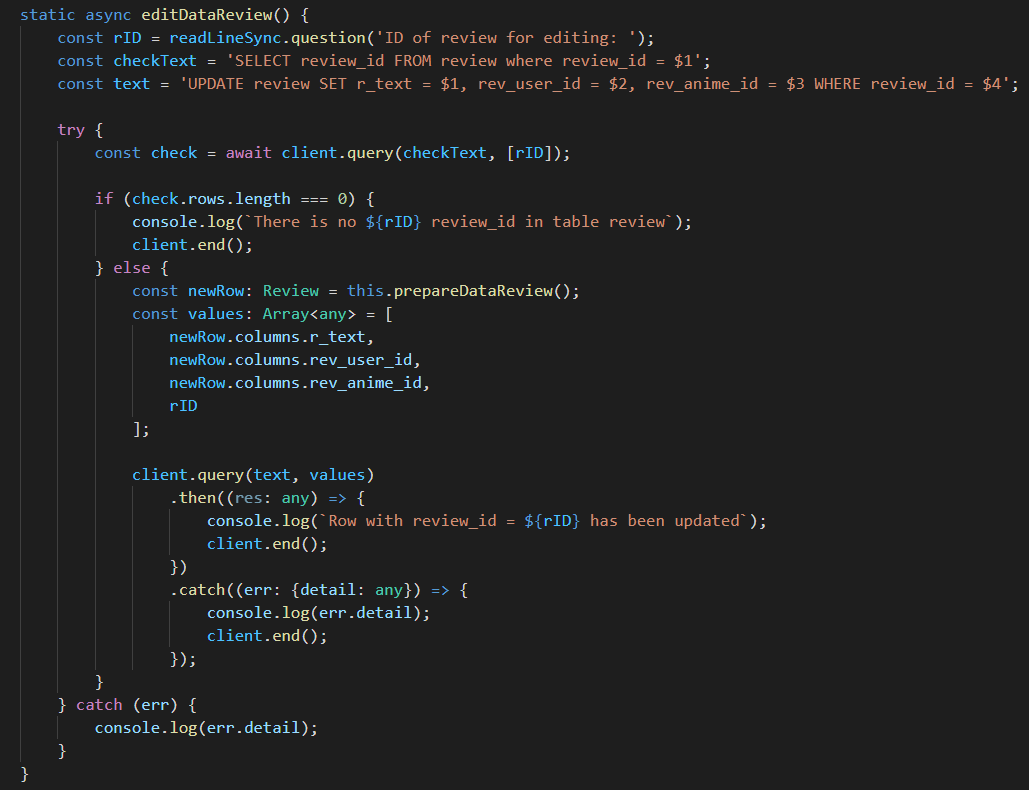


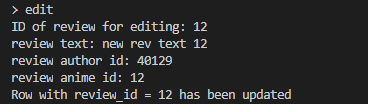


Результат:

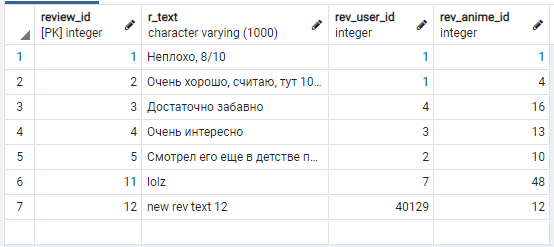


Редагування запису в таблиці review

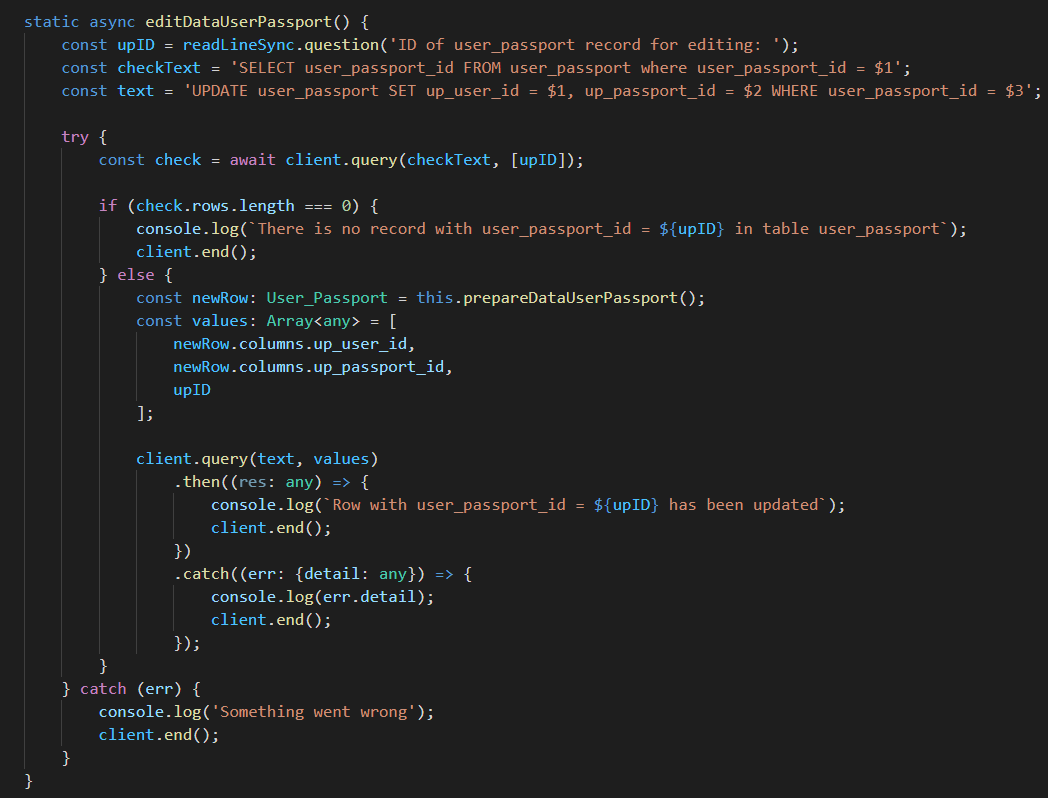


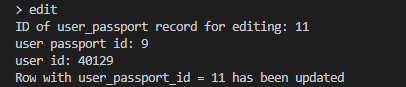


Результат:

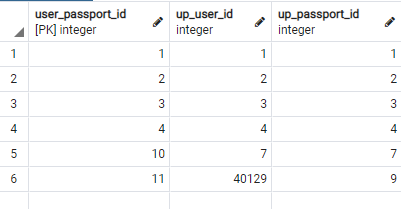


Редагування запису в таблиці user\_passport

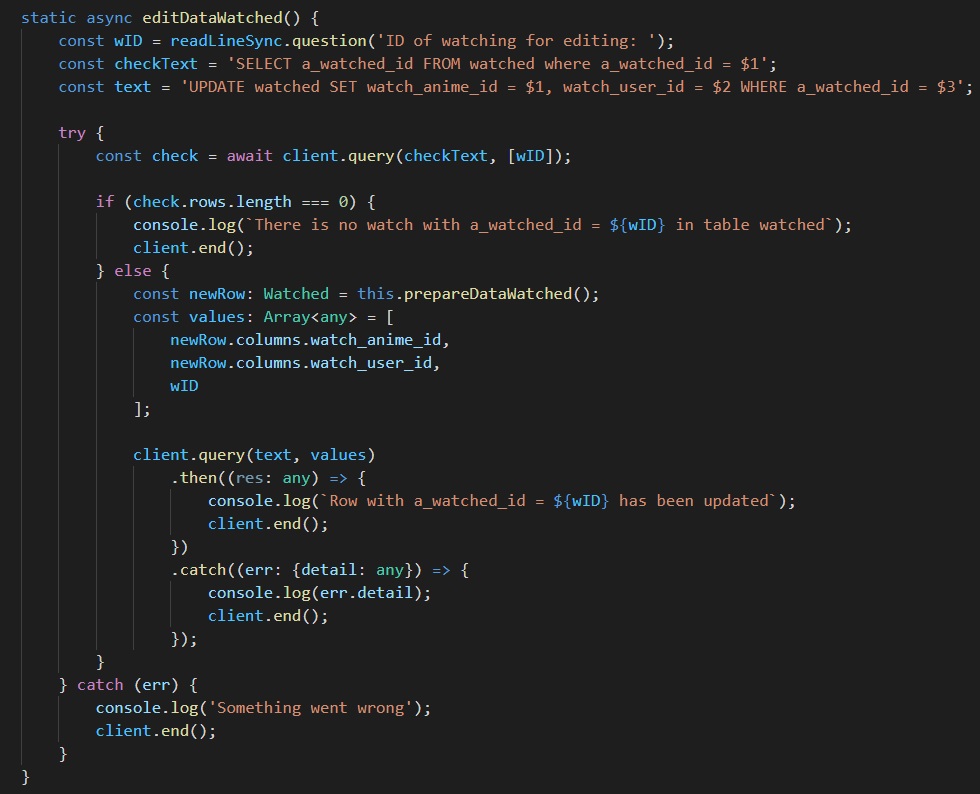


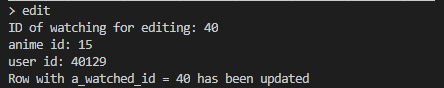


Результат:

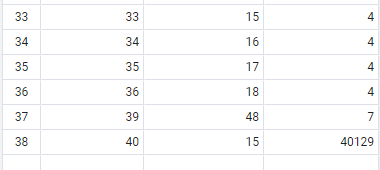


Редагування запису в таблиці watched





Результат:

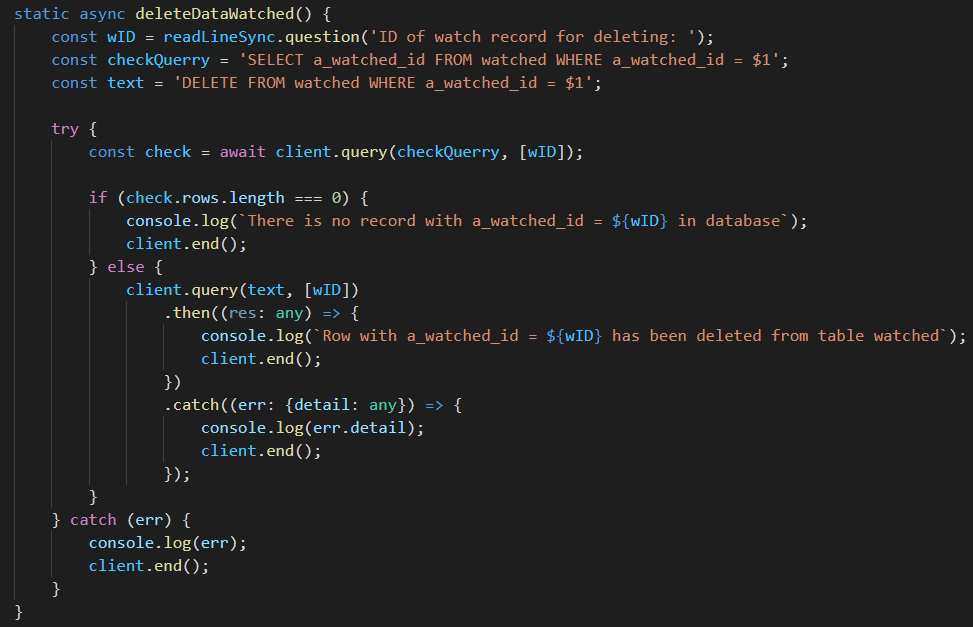


**Функції для видалення**

Видалення відбувається через метод **remove,** що має доступ до змінної, в якій зберігається назва обраної таблиці, за допомогою якої обирається метод видалення для конкретної таблиці:

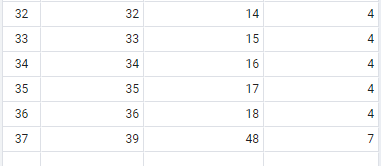


Видалення запису із таблиці watched

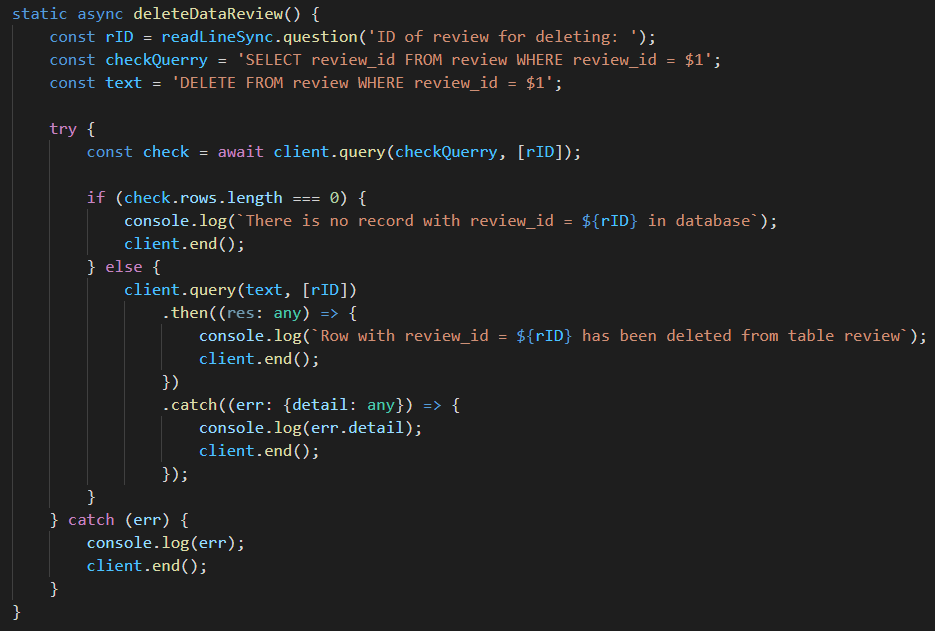




Результат:

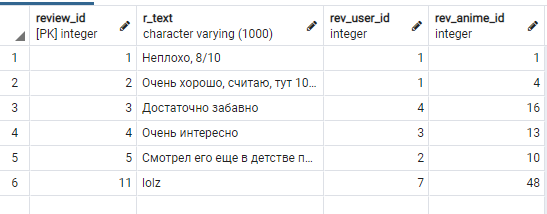


Видалення запису із таблиці review

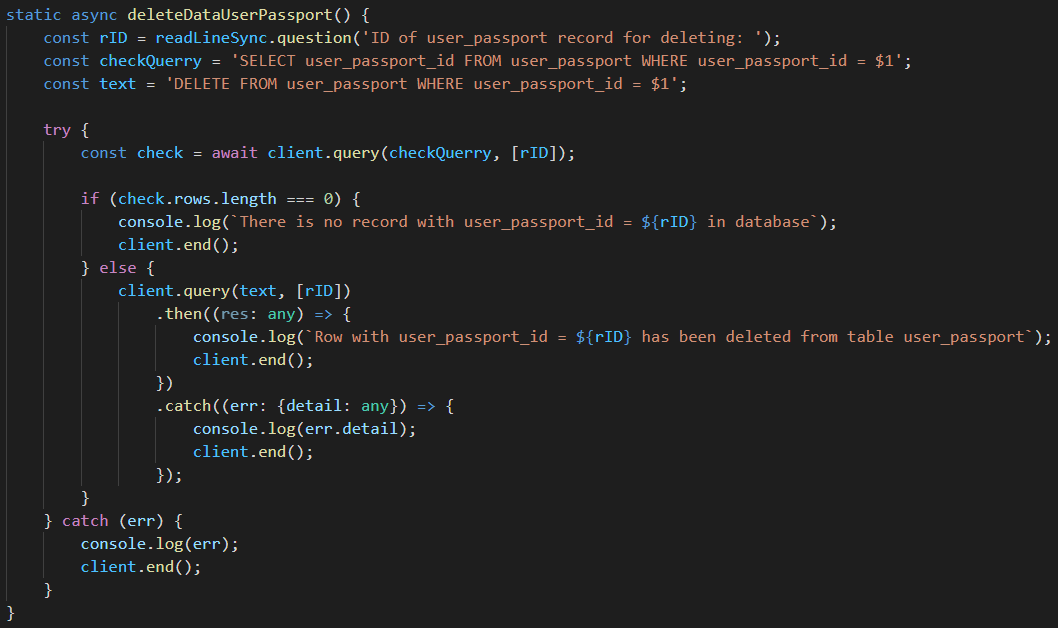




Результат:

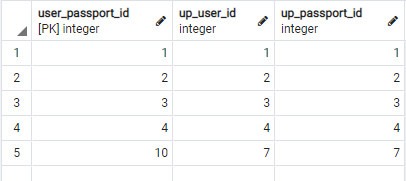


Видалення запису із таблиці user\_passport

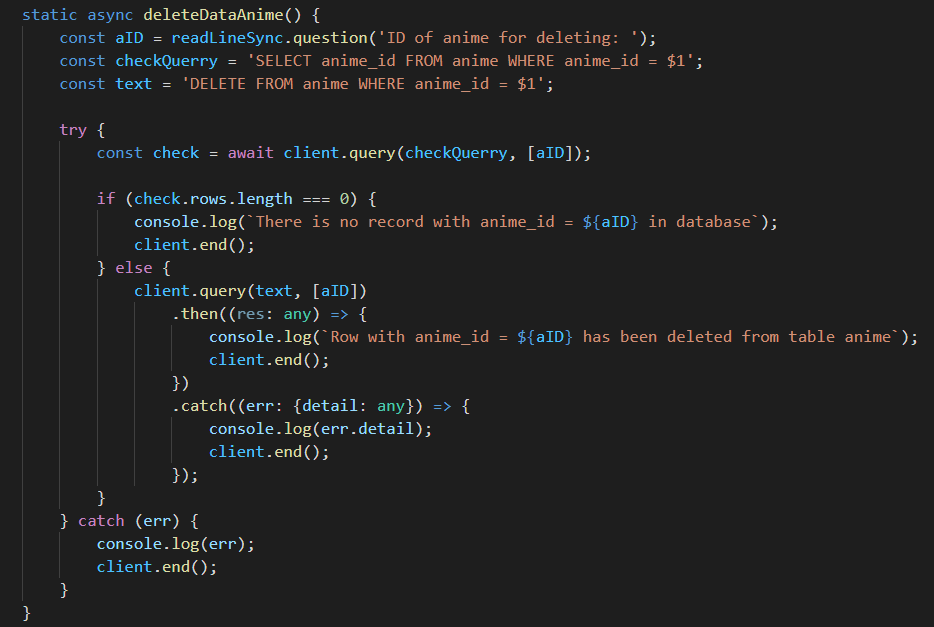




Результати:

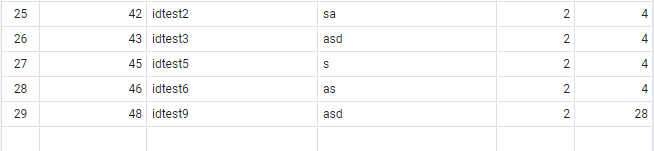


Видалення запису із таблиці anime

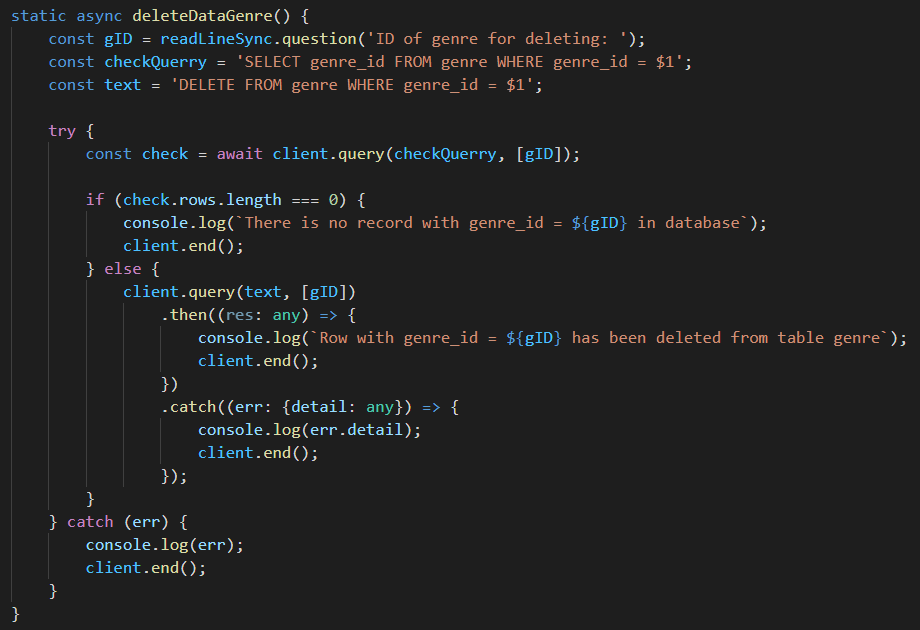




Результат:

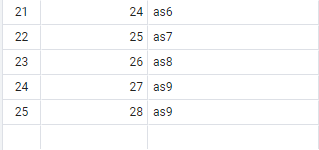


Видалення запису із таблиці genre

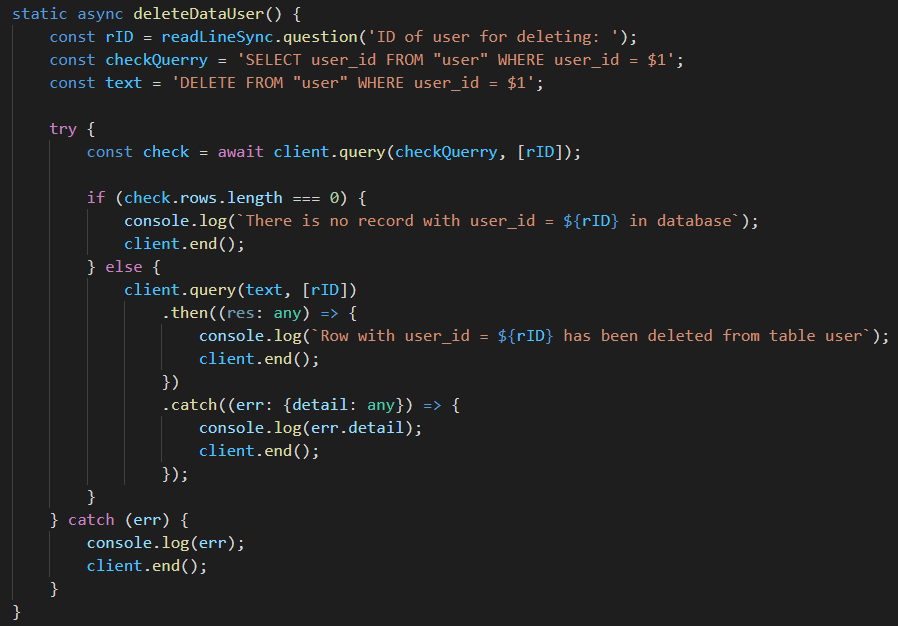




Результат:

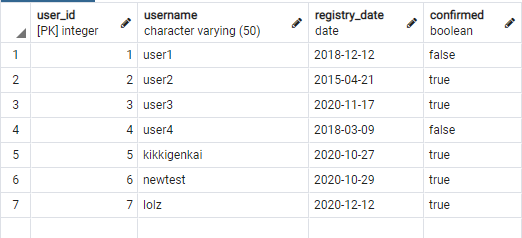


Видалення запису із таблиці user

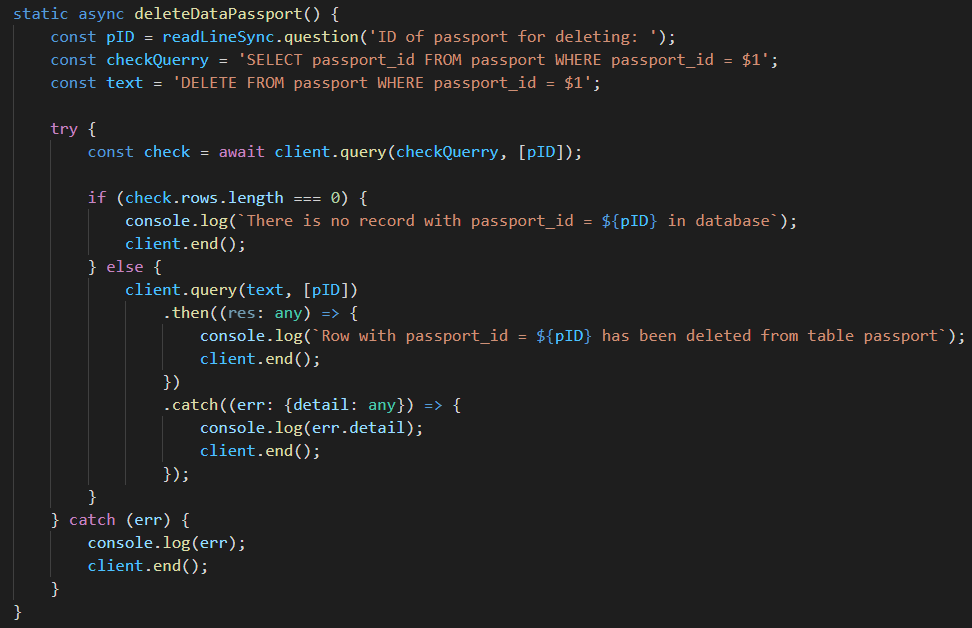




Результати:

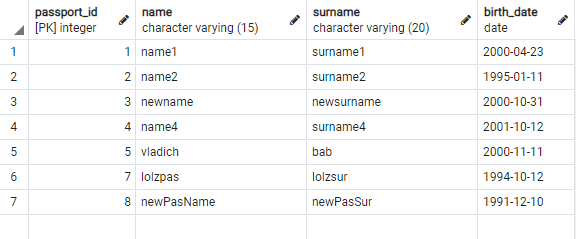


Видалення запису із таблиці passport





Результати:



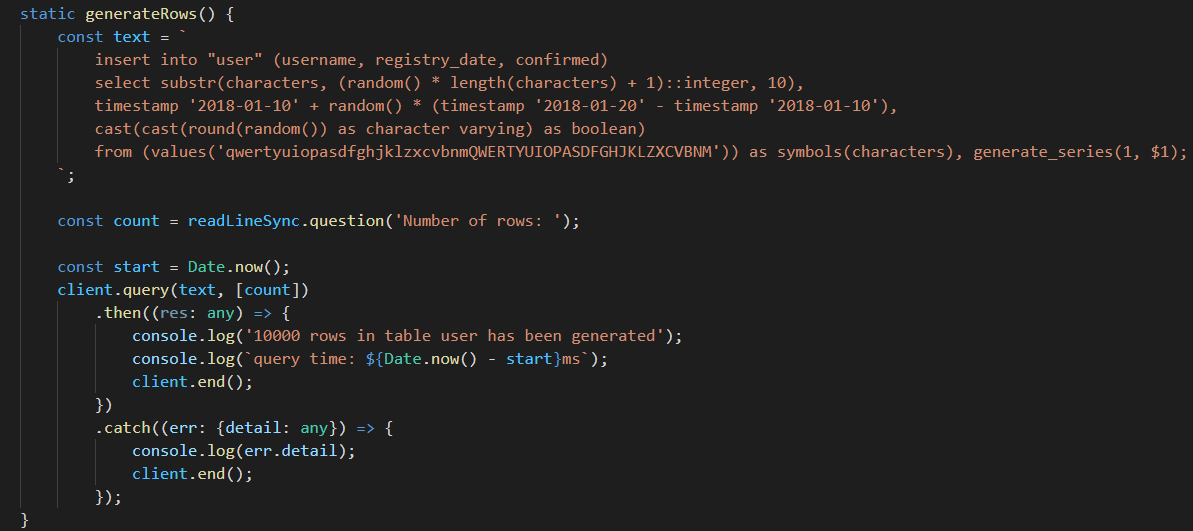
**Лістинги програми з директивами внесення рандомізованих даних**

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

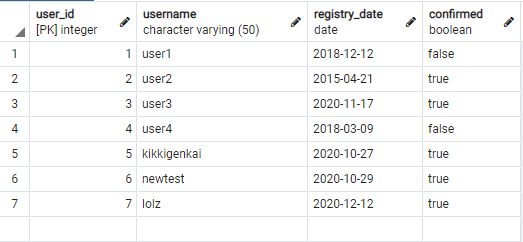
**цих директив**

**Рандомізоване внесення даних до таблиці user**

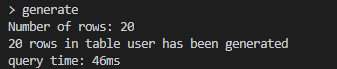
Даний метод генерує задану користувачем кількість рандомізованих даних до таблиці user:

****

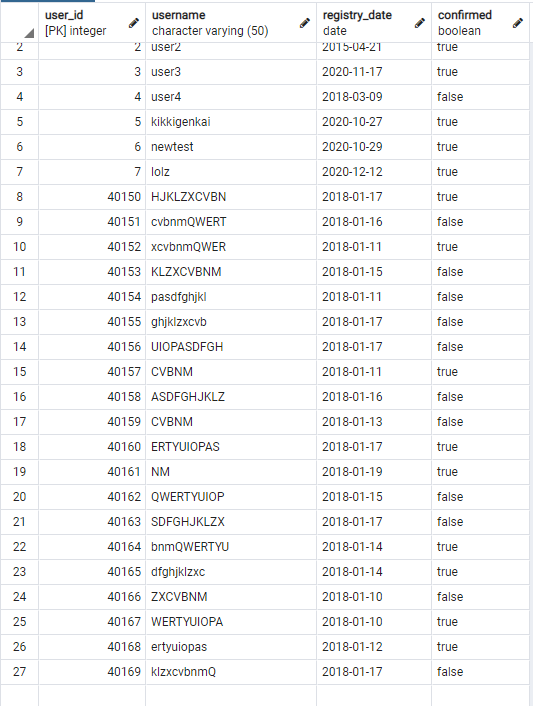
Таблиця user до генерації рядків:

****

Задавання кількості рядків користувачем:

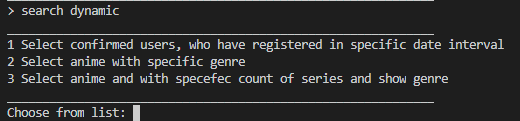


Таблиця user після запиту:



**Виконання динамічних запитів бази даних**

Меню вибору запиту:

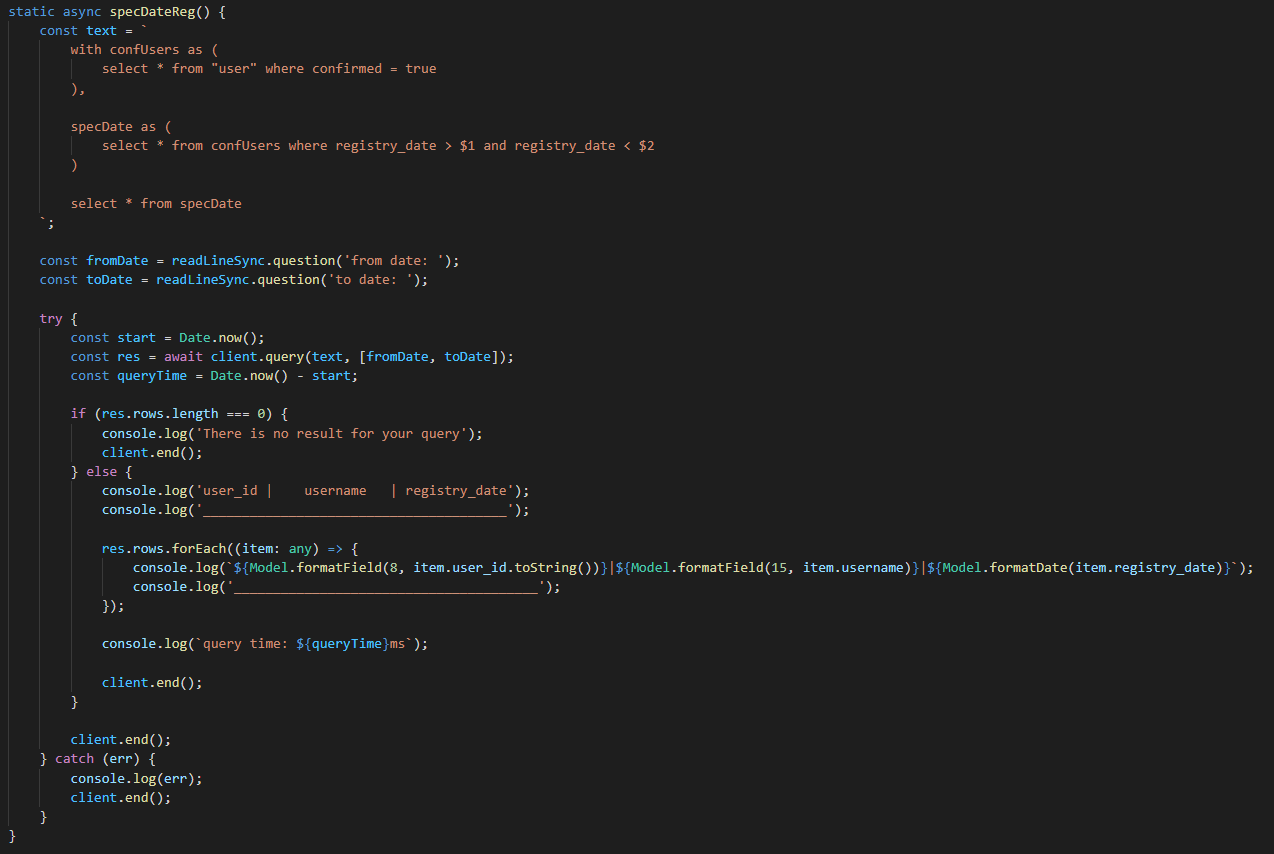
****

Виклик функції з обраним запитом відбувається у контроллері:

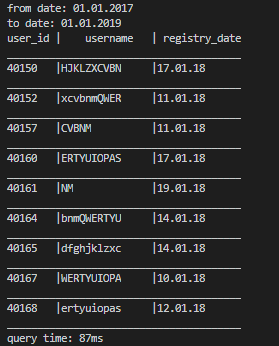


Обрахунок часу запиту відбувається безпосередньо у кожному методі моделі, що обробляє запит. Береться поточний час ДО відсилання запиту до бази даних і виводиться різниця між поточним часом ПІСЛЯ запиту і часом ДО запиту, щоб не враховувати час на форматування виводу у консоль.

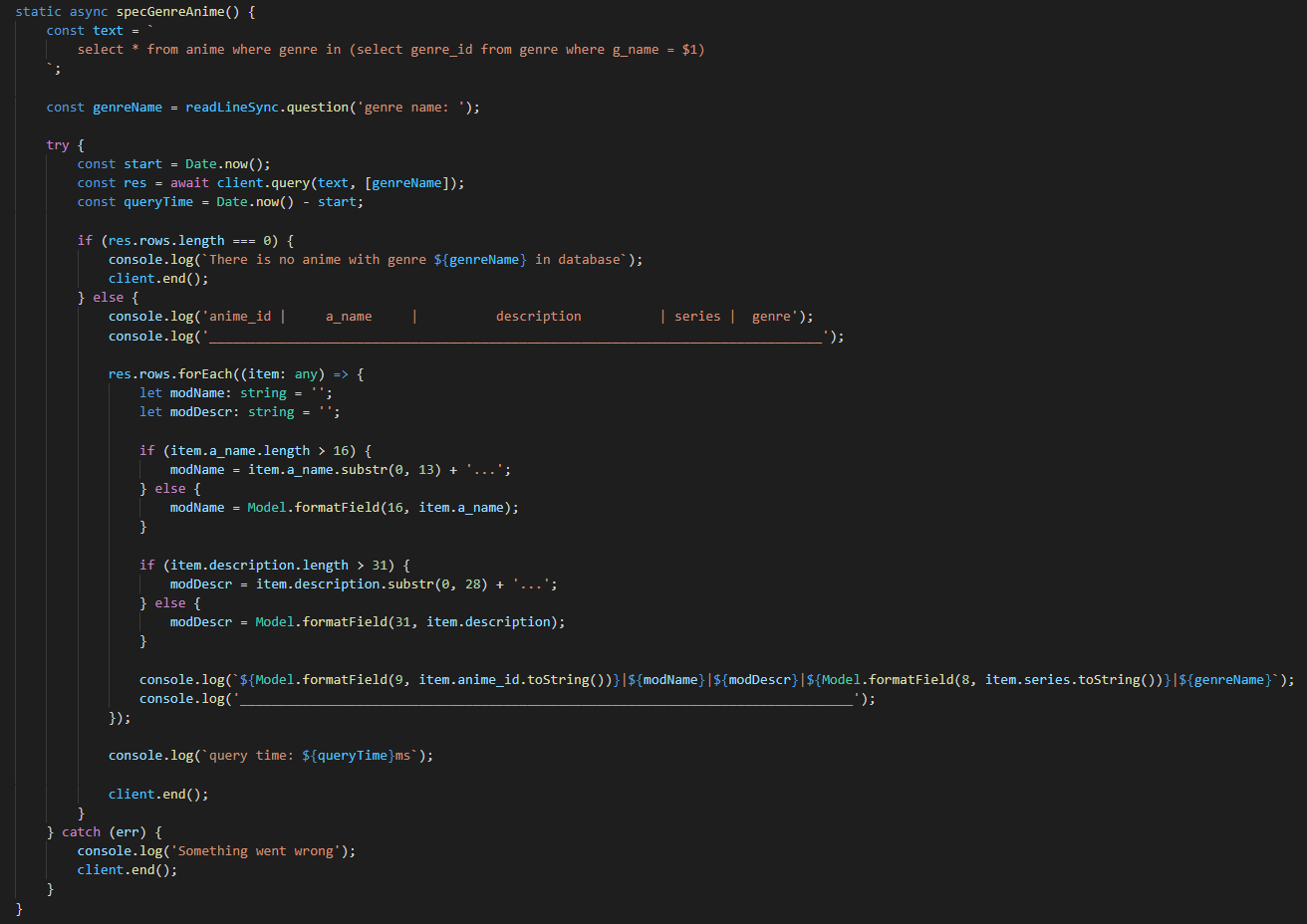
Перший запит:



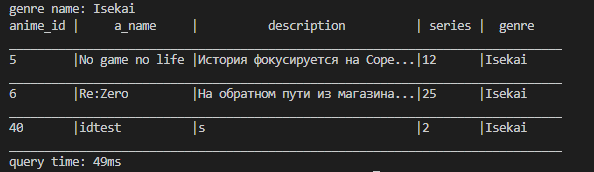
Результат:



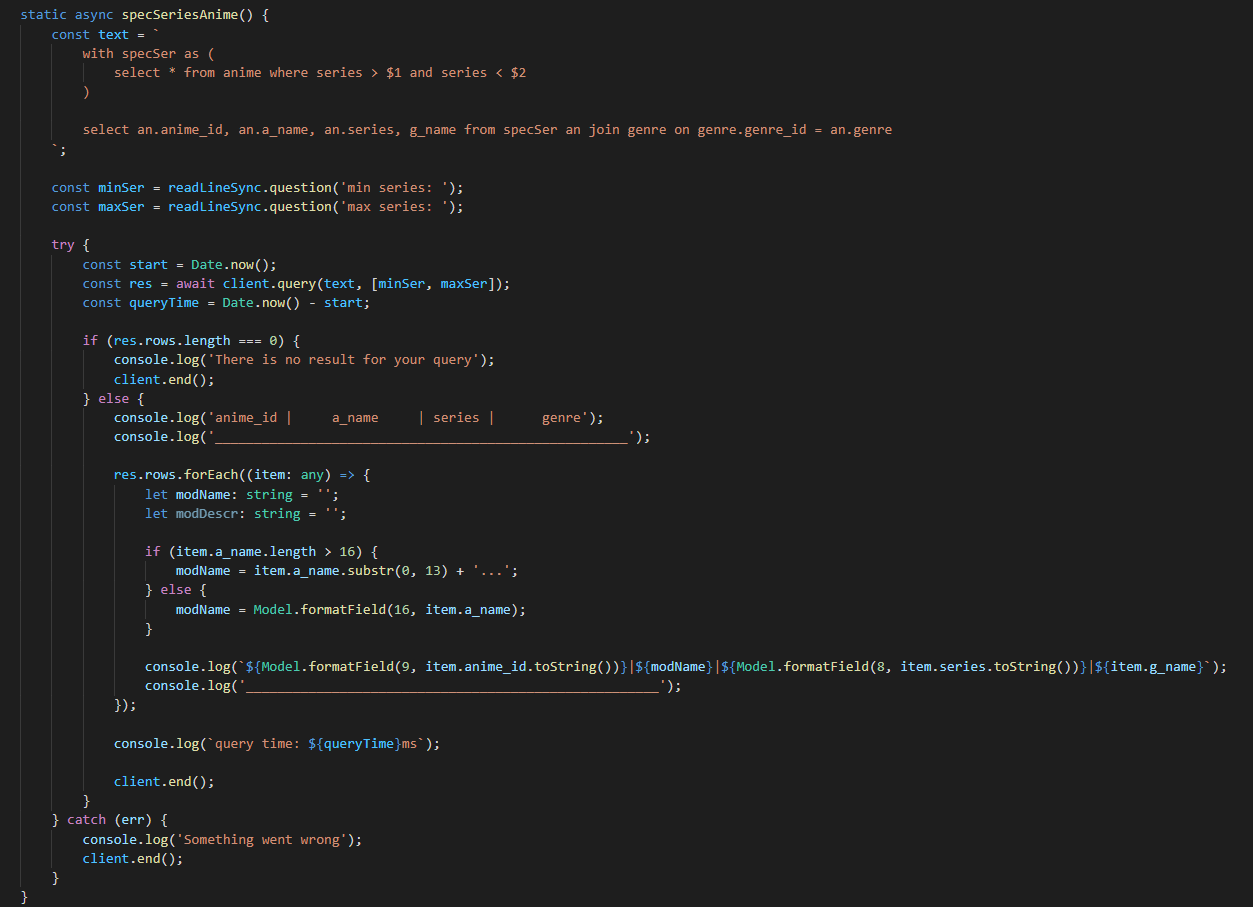
Другий запит:



Результат:



Третій запит:



Результат:

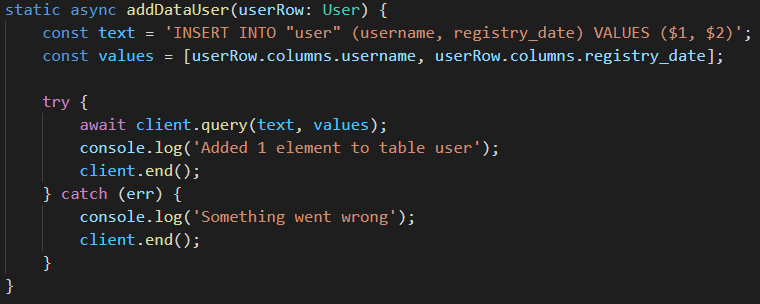


**Обробка виняткових ситуацій (помилок) при введенні/вилученні**

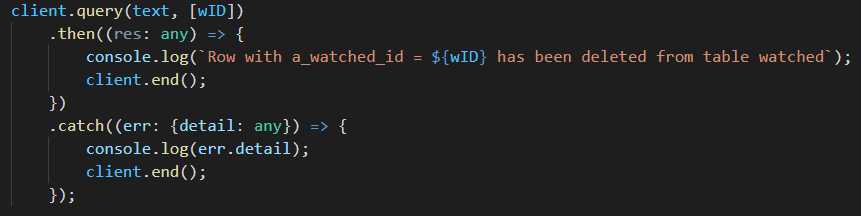
**та валідації даних**

Обробка виняткових ситуацій при введенні та видаленні даних виконується за допомогою блоків try-catch(при використанні async/await підходу) та блоків then/catch(при використанні promise підходу):

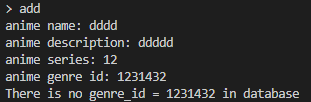
Приклад async/await підходу:



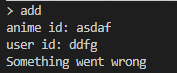
Приклад promise підходу:

****

Додавання рядка із неіснуючим зовнішнім ключем:



Введення рядка з полем, тип якого не відповідає дійсному:



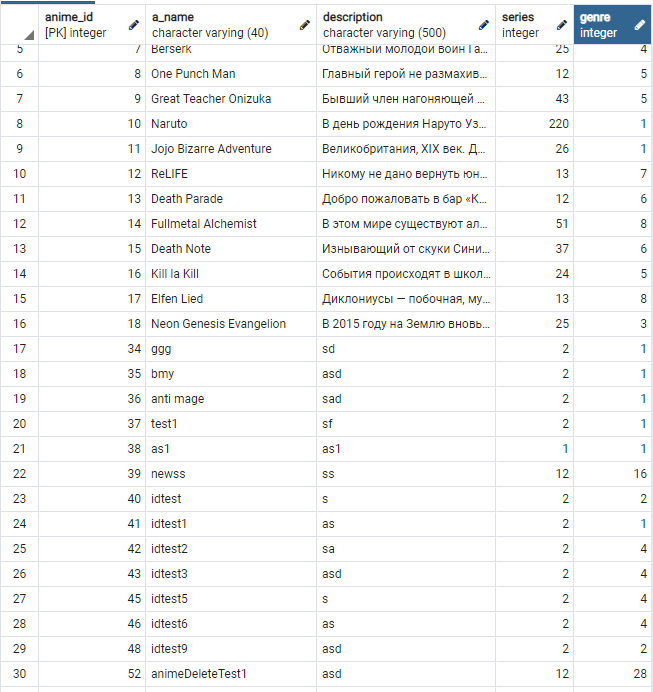
Видалення рядка, ключ якого є зовнішнім ключем іншої таблиці:



**Дослідження режимів обмеження ON DELETE**

Дослідження режимів будемо проводити на таблиці genre (батьківська) та anime (дочірня). Використаємо запис із таблиці anime з anime\_id = 52(в майбутньому запис був видалений та записаний під новим anime\_id = 54) та зовнішнім ключем genre = 28:

Таблиця anime:



Таблиця genre:



Режим NO ACTION

При видаленні запису із таблиці genre, id якого присутній в таблиці anime отримуємо повідомлення про помилку:



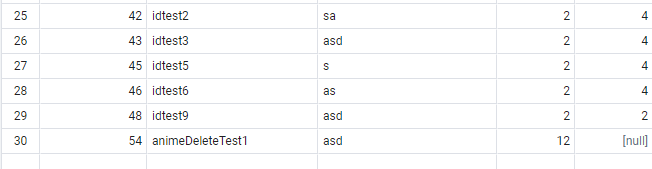
Режим SET NULL

При видаленні запису із таблиці genre, id якого присутній в таблиці anime, якщо на genre таблиці anime NOT NULL, отримуємо повідомлення:



Якщо прибрати NOT NULL:





Запис з таблиці genre було видалено, а поле genre таблиці anime перейшло у null.

Використаємо запис таблиці genre id = 31, g\_name = deleteGenreTest2

Режим SET DEFAULT



В налаштуваннях genre 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 genre, id якого присутній в таблиці anime отримуємо таке саме повідомлення про помилку, як і в режимі NO ACTION:

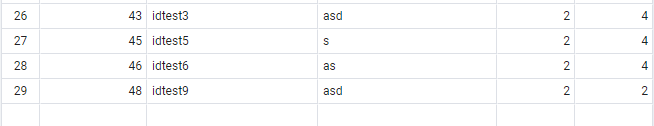


Режим CASCADE

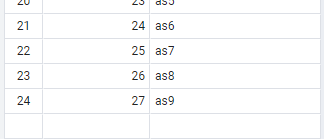
При видаленні запису із таблиці genre, id якого присутній в таблиці anime, видаляється запис таблиці anime та запис таблиці genre із відповідним genre\_id:



Таблиця anime



Таблиця genre



**Ілюстрації програмного коду на Github**

