НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра інформатики та програмної інженерії
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2022 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення комп'ютеризованих систем»

спеціальності «121 Інженерія програмного забезпечення»

на тему: Веб-застосування для побудови запитів до бази даних PostgreSQL із використанням графічного інтерфейсу

Виконав студент IV курсу, групи ІП-81
(шифр групи)

Федоров Олександр Вікторович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник доцент, к.т.н., доц., Ліщук К. І. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з графічної документації доцент, к.т.н., доц., Ліщук К. І. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент доцент кафедри ІСТ, к.т.н., доц., Писаренко А. В. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2022

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 Інженерія програмного забезпечення

Освітньо-професійна програма – Інженерія програмного забезпечення комп'ютеризованих систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2022 р.

ЗАВДАННЯ
на дипломний проєкт студенту

Федорову Олександру Вікторовичу
(прізвище, ім'я, по батькові)

1. Тема проєкту Веб-застосування для побудови запитів до бази даних PostgreSQL із використанням графічного інтерфейсу

керівник проєкту Ліщук Катерина Ігорівна, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «10» червня 2022 р. №1033-с

2. Термін подання студентом проєкту « 19 » червня 2022 року

3. Вихідні дані до проєкту: технічне завдання

4. Зміст пояснювальної записки
1) Аналіз вимог до програмного забезпечення: основні визначення та терміни, опис предметного середовища, огляд існуючих рішень, постановка задачі.
2) Моделювання та конструювання програмного забезпечення: моделювання та аналіз програмного забезпечення, архітектура програмного забезпечення, конструювання програмного забезпечення, аналіз безпеки даних.
3) Аналіз якості та тестування програмного забезпечення.
4) Впровадження та супровід програмного забезпечення.

5. Перелік графічного матеріалу

1) Схема структурна варіантів використань _____

2) Схема структурна компонентів програмного забезпечення _____

3) Креслення вигляду екранних форм _____

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «10» березня 2022 року _____

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	10.03.2022	
2	Аналіз існуючих методів розв’язання задачі	15.03.2022	
3	Постановка та формалізація задачі	20.03.2022	
4	Розробка інформаційного забезпечення	30.03.2022	
5	Алгоритмізація задачі	05.04.2022	
6	Обґрунтування вибору використаних технічних засобів	10.04.2022	
7	Розробка програмного забезпечення	15.05.2022	
8	Налагодження програми	19.05.2022	
9	Виконання графічних документів	21.05.2022	
10	Оформлення пояснювальної записки	01.06.2022	
11	Подання ДП на попередній захист	06.06.2022	
12	Подання ДП рецензенту	12.06.2022	
13	Подання ДП на основний захист	19.06.2022	

Студент _____ Олександр ФЕДОРОВ
(підпис) (ініціали, прізвище)

Керівник _____ Катерина ЛІЩУК
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з чотирьох розділів, містить 38 таблиць, 11 рисунків та 10 джерел – загалом 59 сторінок.

Дипломний проєкт націлений на спрощення процесу роботи із базами даних PostgreSQL шляхом надання графічного інтерфейсу користувача, для побудови найбільш використовуваних видів запитів до баз даних, що дозволить мінімізувати необхідність писати код власноруч і зменшити ймовірність допущення користувачами помилок.

Мета: розробити програмне забезпечення для взаємодії із базами даних, яке мінімізує необхідність написання програмного коду власноруч.

Об'єкт дослідження: програмне забезпечення для роботи із базами даних PostgreSQL.

Предмет дослідження: процеси автоматизації побудови запитів до баз даних і процеси побудови запитів до баз даних шляхом використання графічного інтерфейсу.

У розділі аналізу вимог до програмного забезпечення було наведено аналіз предметної області, розглянуто перелік існуючих аналогічних програмних продуктів і перелічено вимоги до розроблюваної системи.

Розділ моделювання та конструювання програмного забезпечення присвячений моделювання системи, визначенню оптимальної архітектури та функціоналу, аналізу безпеки даних, із котрими та працює.

У розділі аналізу якості та тестування програмного забезпечення було описано елементи системи, котрі мають бути протестовані, надано опис процедур їх тестування та специфічних тест-кейсів, котрі мусять бути виконані.

Розділ впровадження та супроводу програмного забезпечення описує вимоги до середовища, у якому систему можна розгорнути, та процес подальшої її підтримки.

КЛЮЧОВІ СЛОВА: БАЗА ДАНИХ, POSTGRESQL, ГЕНЕРАЦІЯ КОДУ, ГРАФІЧНИЙ ІНТЕРФЕЙС КОРИСТУВАЧА.

ABSTRACT

The explanatory note of the diploma project consists of four sections, contains 38 tables, 11 figures and 10 sources – in total 59 pages.

The diploma project aims to simplify the process of working with databases by providing a graphical user interface for creation of queries that are used the most often which allows to minimize the need in writing code manually and decrease the probability of making mistakes by the users.

The main purpose is to develop a piece of software for database interaction which minimizes the need to manually write code yourself.

The object of the study is software for working with PostgreSQL databases.

The subjects of the study are processes of database query building automation and processes of building database queries with the use of a graphical user interface.

In the requirements analysis section the subject area was analyzed, the existing analogous systems were described, and the requirements for the developed system were listed.

In the software modeling section the system model was created, the optimal architecture and functionality for it were decided upon, and the system security was analyzed.

In the quality assurance section the system elements which have to be tested were described together with overall procedures and specific test cases used for that.

In the deployment and maintenance section the requirements for the deployment environment were described together with the system's further maintenance process.

KEYWORDS: DATABASE, POSTGRESQL, CODE GENERATION, GRAPHICAL USER INTERFACE.

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІП-8135.045300.01.91	Технічне завдання	11	
3	А4	КПІ.ІП-8135.045300.02.81	Пояснювальна записка	59	
4	А4	КПІ.ІП-8135.045300.03.13	Опис програми	34	
5	А4	КПІ.ІП-8135.045300.04.51	Програма та методика тестування	6	
6	А4	КПІ.ІП-8135.045300.05.34	Керівництво користувача	16	
7	А3	КПІ.ІП-8135.045300.06.99	Графічний матеріал	4	
8	А3	КПІ.ІП-8135.045300.06.99СС	Схема структурна варіантів	1	
			використань		
9	А3	КПІ.ІП-8135.045300.06.99СС	Схема структурна компонентів	1	
			програмного забезпечення		
10	А3	КПІ.ІП-8135.045300.06.99КЕ	Креслення вигляду екранних	1	
			форм		

Змін.	Арк.	№ докум.	Підп.	Дата	КПІ.ІП-8135.045300.00.90		
Розроб.		Федоров О.В.			Відомість дипломного проєкту	Літ.	Аркуш 1 / Аркушів
Перевір.		Ліщук К.І.				КПІ ім. Ігоря Сікорського ФІОТ каф. ІПІ гр. ІП-81	
Н.контр.		Ліщук К.І.					
Затв.		Жаріков Е.В.					

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“___” _____ 2022 р.

Веб-застосування для побудови запитів до бази даних PostgreSQL із використанням графічного інтерфейсу

Технічне завдання

КПІ.ІП-8135.045300.01.91

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Катерина ЛІЩУК

Нормоконтроль:

_____ Катерина ЛІЩУК

Виконавець:

_____ Олександр ФЕДОРОВ

Київ – 2022

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Веб-застосування для побудови запитів до бази даних PostgreSQL із використанням графічного інтерфейсу.

Галузь застосування:

Наведене технічне завдання поширюється на розробку «Веб-застосування для побудови запитів до бази даних PostgreSQL із використанням графічного інтерфейсу», котра використовується для побудови запитів до баз даних PostgreSQL та призначена для мінімізації необхідності написання SQL коду власноруч серед розробників програмного забезпечення, котрі працюють із базами даних, та людей, котрі вивчають мову SQL.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки «Веб-застосування для побудови запитів до бази даних PostgreSQL із використанням графічного інтерфейсу» є завдання на дипломне проєктування, затверджене кафедрою інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для спрощення процесу роботи із базами даних PostgreSQL.

Метою розробки є надання графічного інтерфейсу для побудови запитів до баз даних, що дозволяє мінімізувати необхідність писати код власноруч і зменшити ймовірність допущенням користувачами помилок.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

4.1.1 Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1.1 Для користувача:

– створення таблиць;
– редагування таблиць;
– імпортування таблиць;
– експортування таблиць;
– видалення таблиць;
– побудова запитів;
– валідація коду запиту;
– копіювання коду запиту.

4.1.2 Розробку виконати у формі клієнтського застосування, серверного застосування і бази даних.

4.1.3 Додаткові вимоги:

Додаткові вимоги відсутні.

4.2 Вимоги до надійності

4.2.1 Передбачити контроль введення інформації.

4.2.2 Передбачити захист від некоректних дій користувача.

4.2.3 Забезпечити неможливість користувачу вплинути на стан серверної бази даних.

4.3 Умови експлуатації

4.3.1 Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.2 Обслуговування – вимоги не висуваються.

4.3.3 Обслуговуючий персонал – вимоги не висуваються.

4.4 Вимоги до складу і параметрів технічних засобів

4.4.1 Програмне забезпечення повинно функціонувати на комп'ютерах із операційною системою на базі Linux та Unix.

4.4.2 Мінімальна конфігурація клієнтських технічних засобів:

4.4.2.1 Тип процесору ..... Pentium.

4.4.2.2 Об'єм ОЗП ..... 1 Гб.

4.4.2.3 Під'єднання до мережі Інтернет зі швидкістю ..... від 1 Мбіт/с.

4.4.3 Мінімальна конфігурація серверних технічних засобів:

4.4.3.1 Тип процесору ..... Pentium.

4.4.3.2 Об'єм ОЗП ..... 1 Гб.

4.4.3.3 Під'єднання до мережі Інтернет зі швидкістю ..... від 1 Мбіт/с.

4.5 Вимоги до інформаційної та програмної сумісності

4.5.1 Програмне забезпечення повинно працювати під управлінням операційних систем на базі Linux та Unix. Клієнтське застосування повинно працювати в браузерах Google Chrome, Mozilla Firefox і Edge.

4.5.2 Вхідні дані повинні бути представлені в наступному форматі: текстові файли у форматі JSON, HTTP запити із тілом у форматі JSON.

4.5.3 Результати повинні бути представлені в наступному форматі: текстові файли у форматі JSON, HTTP відповіді із тілом у форматі JSON.

4.5.4 Клієнтське застосування повинно бути розроблено мовами HTML, CSS і JavaScript. Серверне застосування повинно бути розроблено

мовою Python 3 із використанням фреймворку Flask. У ролі СУБД має бути використано PostgreSQL.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не пред‘являються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не пред'являються.

4.8 Спеціальні вимоги

Згенерувати установчу версію програмного забезпечення.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

5.2 Програмне забезпечення повинно мати довідникову систему.

5.3 У склад супроводжувальної документації повинні входити наступні документи:

5.3.1 Пояснювальна записка не менше ніж на 50 аркушах формату А4 (без додатків 5.3.2 - 5.3.4).

5.3.2 Технічне завдання.

5.3.3 Керівництво користувача.

5.3.4 Програма та методика тестування.

5.4 Графічна частина повинна бути виконана на 3 аркушах формату А3, котрі включаються у якості додатків до пояснювальної записки:

5.4.1 Схема структурна варіантів використання.

5.4.2 Схема структурна компонентів системи.

5.4.3 Креслення вигляду екранних форм.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

Таблиця 6.1 – Стадії і етапи розробки

№	Назва етапу	Строк	Звітність
1.	Вивчення рекомендованої літератури	10.03	
2.	Аналіз існуючих методів розв’язання задачі	15.03	Опис існуючих рішень
3.	Постановка та формалізація задачі	20.03	Специфікації програмного забезпечення
4.	Розробка інформаційного забезпечення	30.03	Схема структурна компонентів системи
5.	Алгоритмізація задачі	05.04	
6.	Обґрунтування вибору використаних технічних засобів	10.04	
7.	Розробка програмного забезпечення	15.05	Тексти програмного забезпечення
8.	Налагодження програми	19.05	Тести, результати тестування
9.	Виконання графічних документів	21.05	Графічний матеріал проекту
10.	Оформлення пояснювальної записки	01.06	Текст пояснювальної записки
11.	Подання ДП на попередній захист	06.06	
12.	Подання ДП рецензенту	12.06	
13.	Подання ДП на основний захист	19.06	

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Пояснювальна записка
до дипломного проєкту

на тему: Веб-застосування для побудови запитів до бази даних PostgreSQL із використанням графічного інтерфейсу

Київ – 2022

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

SQL	–	Structured Query Language – мова структурованих запитів.
ПЗ	–	Програмне забезпечення.
IDE	–	Integrated Development Environment – інтегроване середовище розробки.
БД	–	База даних.
ІТ	–	Інформаційні технології.
HTML	–	HyperText Markup Language – мова розмітки гіпертексту.
CSS	–	Cascading Style Sheets – каскадні таблиці стилів.
HTTPS	–	Hypertext Transfer Protocol Secure – протокол передачі гіпертекстових документів безпечний.
TLS	–	Transport Layer Security – захист на транспортному рівні.
UML	–	Unified Modeling Language – уніфікована мова моделювання.
СУБД	–	Система управління базами даних.
ПК	–	Персональний комп’ютер.

ВСТУП

На сьогоднішній день велика кількість програмного забезпечення та застосувань взаємодіють із базами даних. Мова програмування SQL була створена 1974 року саме з цією метою. Хоча зараз усе більш популярними стають інструменти, які дозволяють абстрагуватися від її використання, із нею все одно доводиться працювати майже всюди: як під час розробки та застосування ПЗ – для збору, обробки, аналізу та збереження інформації, так і у навчанні – вивчення мови SQL і досі є невід'ємною частиною курсу підготовки розробників ПЗ.

Незважаючи на вищенаведені факти та відносну простоту мови SQL, існує мало інструментів, які б дозволили зменшити необхідність написання коду власноруч, а відповідно й полегшити її використання. Такі застосування не є новими, але в них часто зустрічаються проблеми, які роблять дані рішення незручними для використання, наприклад: наявний функціонал може бути завідомо дуже обмеженим; формат застосунку може сильно ускладнювати написання запитів зі зростанням їх складності; деякі рішення виконані у формі плагінів для IDE, а отже їх неможливо використовувати за відсутності певного програмного забезпечення на комп'ютері користувача.

Із високою ймовірністю, ця ситуація не буде суттєво змінюватися з двох причин, які криються в особливостях самої мови та сучасних тенденціях її використання. По-перше, мова SQL є досить простою і код, який доводиться писати розробникам, також часто є нескладним. Звідси походить думка, що інструменти генерації запитів до БД не є необхідними, оскільки вони намагаються спростити те, що «не потребує подальшого спрощення». Але дане твердження ігнорує той факт, що не кожен розробник-новачок зможе опанувати нову для себе технологію без допоміжних інструментів; крім того є багато людей, які працюють із даними, але не є розробниками ПЗ, наприклад, бізнес-аналітики, і далеко не кожен із них знається на програмуванні достатньо добре, аби власноруч писати запити до баз даних.

По-друге, активно розробляються інструменти, які дозволяють абстрагуватися від взаємодії із БД напряму а також надають спрощений або уніфікований інтерфейс для роботи із різними джерелами даних. Через їх зростаючу популярність, можна почути думку, що мова SQL «вмирає», і відповідно в інструментах генерації запитів для неї скоро не буде потреби взагалі. Звісно, дане твердження не є об’єктивними, оскільки мова SQL нікуди не зникне – вона завжди залишатиметься частиною багатьох існуючих програмних рішень й не скоро зникне із нового ПЗ.

На основі даної аналітики можна зробити висновок, що тема інструментів для генерації коду мовою SQL й надалі залишатиметься актуальною, а розроблюване в рамках даної дипломної роботи програмне рішення знайде своє застосування щонайменш у двох сферах:

– освіти – як інструмент, який допомагає в опануванні мови SQL;

– розробки ПЗ – як інструмент, який спрощує процес написання запитів до баз даних розробникам-професіоналам.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

Веб-застосування для побудови запитів до бази даних PostgreSQL із використанням графічного інтерфейсу – програмне рішення, яке дозволяє користувачам будувати запити мовою SQL із мінімізацією потреби писати код власноруч. Основними його функціями є: надання користувачу інтерфейсу для побудови запитів, надання користувачу інтерфейсу для створення мінімалістичних таблиць із метою подальшої валідації коду, генерація коду мовою SQL на основі наданих користувачем даних, валідація згенерованого коду із використанням робочої бази даних.

1.2 Змістовний опис і аналіз предметної області

Мова SQL є однією із найбільш використовуваних мов програмування, оскільки лише вона дозволяє напряму взаємодіяти із реляційними БД, котрі найчастіше впроваджуються в якості сховища даних. Володіння мовою SQL являється одним із ключових вмінь для сучасних розробників ПЗ, але нерідко нею доводиться користуватися людям, котрі її не знають: розробникам-початківцям, котрі лише вивчають мову SQL; досвідченим розробникам, котрі попередньо не працювали із БД; рідше фахівцям, які не є розробниками ПЗ, але чия сфера діяльності пов’язана із роботою із базами даних – аналітики, бухгалтери тощо. Отже, існує група людей, котрі або не можуть власноруч використовувати мову SQL, або, принаймні, не можуть робити цього в повну силу – зрозуміло, що в такому випадку може стати в нагоді набір інструментів, котрі могли б полегшити процес написання запитів до БД шляхом мінімізації роботи, яку потрібно виконати користувачу, а саме: надавати користувачу простий інтерфейс із переліком найчастіше вживаного функціоналу мови SQL, за допомогою якого потрібно лише задати перелік даних та операцій, котрі мусять бути використані та виконані. Наразі існує не

одне програмне рішення, котре намагається вирішити цю проблему, але через те, що потенційні користувачі сильно відрізняються один від одного, вони часто є спеціалізованими під одну конкретну аудиторію, що робить їх непідходящими для більш ніж однієї сфери застосування. Продукти створені для сфери освіти часто є занадто примітивними для серйозної роботи або навіть самого вивчення мови; для професійних розробників – занадто громіздкі у простих ситуаціях; для інших фахівців – можуть містити переважно високорівневий та абстрактний функціонал, не надаючи користувачу можливість побачити, що саме відбувається в базі даних. Для останньої групи було розроблено багато різноманітних програмних продуктів для будь-яких можливих потреб, але ситуація із продуктами, орієнтованими на розробників ПЗ, значно інша – застосування, котрі були б достатньо просунуті для глибокого вивчення мови SQL і використання в роботі а також котрі залишаються відносно простими у використанні є рідкістю.

1.3 Аналіз успішних IT-проектів

1.3.1 Аналіз відомих технічних рішень

Існує немала кількість програмних продуктів, котрі націлені на спрощення написання запитів до баз даних. Більшість із них суттєво відрізняються між собою форматом підтримуваного функціоналу, форматом застосунку та способом його встановлення або впровадження. Якщо розглянути кожну із цих характеристик окремо, то можна виокремити наступні варіанти того, які підходи використовуються при розробці схожих програмних рішень:

а) за форматом підтримуваного функціоналу:

– низькорівневі – із можливістю напряму використовувати фактичні можливості мови SQL;

– високорівневі – із використанням шаблонів та макросів, котрі потребують мінімальну кількість інформації від користувача, аби будувати комплексні запити;

б) за форматом застосунку:

– переважно текстові – застосування, в яких інтерфейс користувача реалізований у вигляді форм із текстовими полями, спадними списками, кнопками тощо;

– переважно графічні – застосування, в яких інтерфейс користувача реалізований у вигляді поля, де розміщуються вузли, які об’єднуються в єдиний граф;

в) за способом встановлення або впровадження:

– веб-застосування – застосування, які використовують інтерфейс веб-браузера;

– самостійні програми – застосування, які потребують розміщення або встановлення на комп’ютері користувача;

– плагіни – застосування, які виступають в якості розширень для іншого програмного забезпечення.

1.3.2 Аналіз відомих програмних продуктів

Задля аналізу існуючих програмних продуктів розглянемо програмні продукти відповідно до аудиторій користувачів, на які вони орієнтовані.

1.3.2.1 Програмні продукти орієнтовані на сферу освіти

У якості прикладу можна навести інструмент під назвою BlocklySQL, показаний на рисунку 1.1.

Рисунок 1.1 – Зовнішній вигляд блоків, що реалізують запит до бази даних, у застосуванні BlocklySQL

BlocklySQL є інструментом візуального програмування на основі блоків – користувачі можуть обирати елементи, які відповідають за виконання певних операцій, і об’єднувати їх між собою у єдину структуру. Він був розроблений у Вюрцбурзькому університеті, після проведення дослідження того, яких помилок найчастіше допускають студенти під час вивчення мови SQL, аби допомогти їм у навчанні.

Перевагою такого продукту є те, що користувачу взагалі не потрібно писати код, тому ймовірність здійснення помилок мінімальна. Але така надмірна безпека та простота у використанні досягається шляхом суттєвого обмеження підтримуваного функціоналу, тому будувати комплексні запити часто дуже важко або навіть неможливо.

1.3.2.2 Програмні продукти орієнтовані на професійних розробників

У якості прикладу можна навести застосування FlySpeed SQL Query, показане на рисунку 1.2.

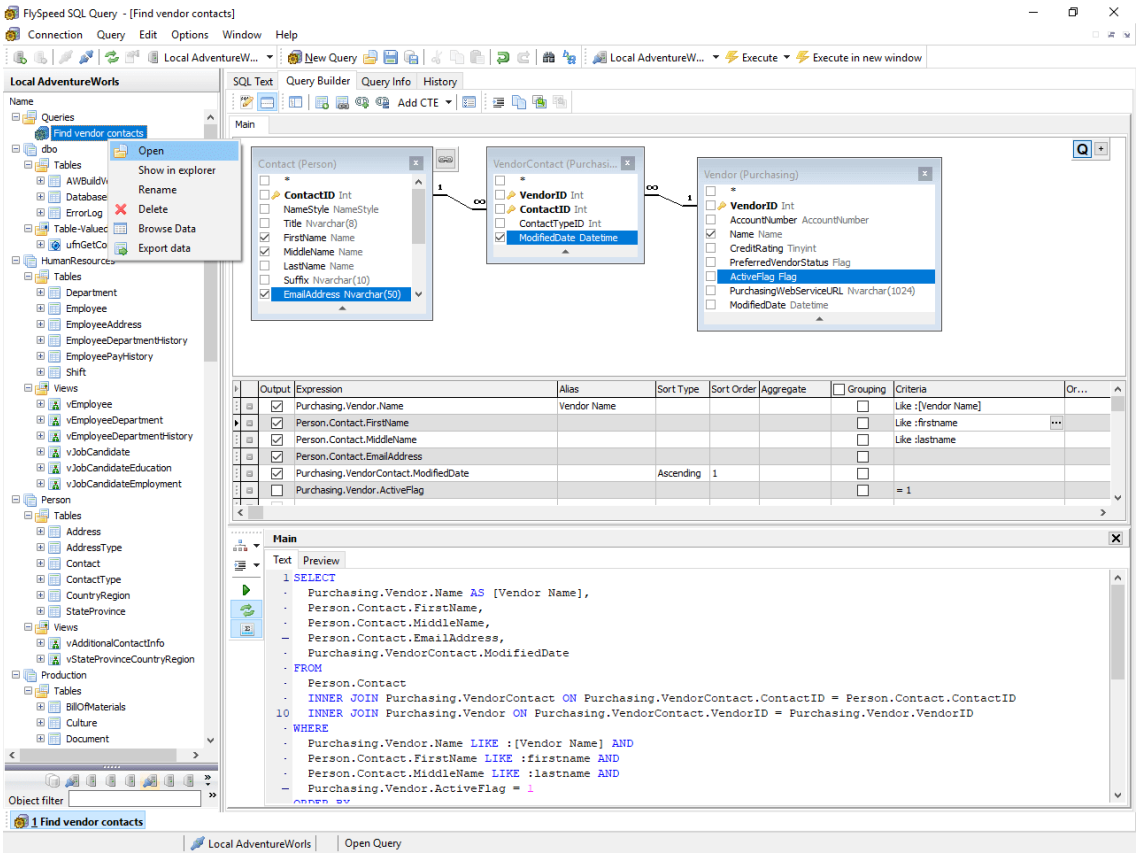


Рисунок 1.2 – Інтерфейс продукту FlySpeed SQL Query під час роботи

FlySpeed SQL Query – застосування, виконане у формі візуального редактора, що є типовим серед інструментів, створених для професійної роботи із базами даних. Користувач може розміщувати вузли, котрі відповідають таблицям БД, з'єднувати їх між собою ланцюгами та задавати будь-які інші операції, що мають бути виконані, у спеціалізованих формах.

Перевагою таких продуктів є те, що вони мають багатий інтерфейс, який дозволяє використовувати більшість можливостей мови SQL. Звісно, такий низькорівневий підхід робить їх погано пристосованими для людей, котрі не є розробниками ПЗ, через свою відносну складність. Серед недоліків слід згадати той факт, що такі програми практично завжди мають громіздкий інтерфейс користувача: із зростанням кількості таблиць, які використовуються в запиті, поле візуального редактора захаращується настільки, що в ньому стає складно орієнтуватися. Крім того, такі продукти майже завжди є комерційними, що робить їх недоступними для широкого загалу.

1.3.2.3 Програмні продукти орієнтовані на фахівців-нерозробників ПЗ

У якості прикладу можна навести застосування Chartio Visual SQL, показане на рисунку 1.3.

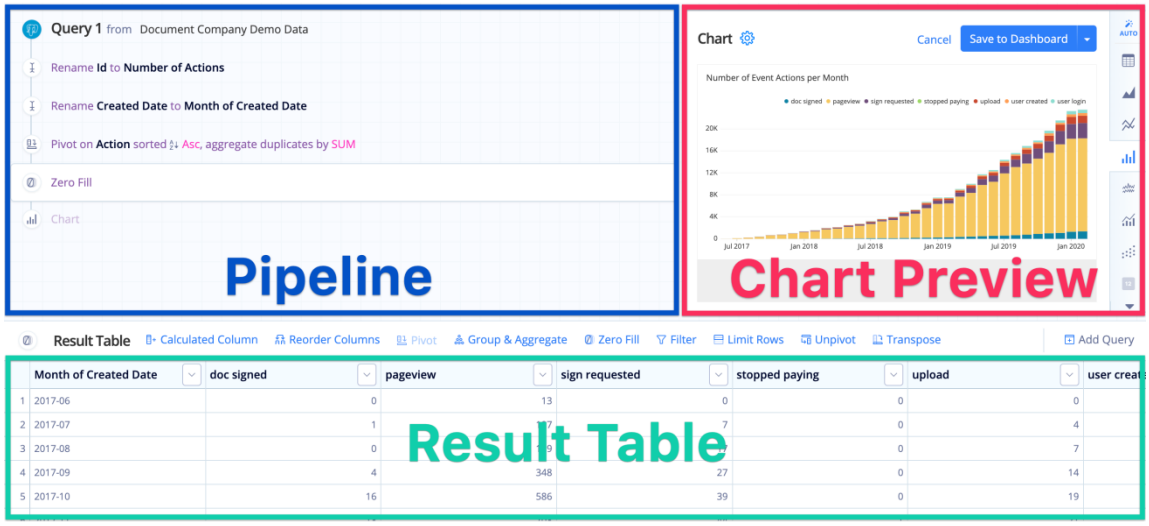


Рисунок 1.3 – Елементи інтерфейсу застосування Chartio Visual SQL

Chartio Visual SQL – BI-застосування, яке підтримує набір спеціалізованих високорівневих операцій, котрі користувач може використати, для взаємодії із базою даних без жодної згадки SQL коду.

Перевагою таких продуктів є те, що вони мають дуже простий та зрозумілий інтерфейс для фахівців, які взагалі не знаються на мові SQL. Вони можуть мати вбудований набір високорівневих функцій, які відповідають за виконання комплексних операцій, що робить такі застосування особливо зручними. Але така простота інтерфейсу досягається обмеженням функціоналу мови SQL доступного користувачу; створення комплексних запитів може бути повільним через необхідність постійно перемикатися між різними редакторами елементів запиту; самі ці редактори можуть мати обов’язкові валідатори, котрі не дозволяють тимчасово задавати лише частину необхідної інформації для генерації запиту; в залежності від застосунку, функція перегляду згенерованого SQL коду може бути взагалі відсутньою.

1.4 Аналіз вимог до програмного забезпечення

1.4.1 Розроблення функціональних вимог

Перед тим як визначати, які функції мають підтримуватися розроблюваним застосуванням, слід також визначити, хто є його користувачами та які існують глобальні його складові.

У даному застосуванні не передбачено поділу користувачів на різні ролі. Потреба в інструментах автентифікації чи авторизації користувачів також відсутня, оскільки весь наявний функціонал системи є доступним для кожного.

Існує дві ключові складові, із якими будуть пов’язані все функції системи: підсистема роботи із таблицями та підсистема роботи із запитами. Перша підсистема дозволяє працювати із «таблицями» – об’єктами, котрі являються мінімалістичною абстракцією над справжніми таблицями бази даних. Вони не мають впливу на процес побудови запитів але використовуються для валідації згенерованого коду. Друга підсистема дозволяє працювати над запитами – за допомогою спеціалізованого інтерфейсу користувач може будувати запити до баз даних, які будуть представлені у вигляді програмного коду мовою SQL.

На рисунку документа КПІ.ІП-8135.045300.06.99СС зображено схему структурну варіантів використання розроблюваного застосування, а доступні користувачу варіанти використання серед них описано в таблицях 1.1-1.8.

Таблиця 1.1 – Варіант використання UC01

Назва	Створення таблиці
Опис	Користувач може створити нову таблицю, котра буде використовуватися для валідація запиту
Передумови	–
Постумови	Додано нову таблицю

Продовження таблиці 1.1

Основний сценарій	1) Користувач натискає на кнопку «Add table». 2) Відкривається вікно створення таблиці. 3) Користувач задає інформацію про таблицю шляхом вводу її назви та додавання набору полів також задаючи їх назви, типи даних і розмірності. 4) Користувач натискає на кнопку «Save». 5) Система валідує введені дані. 6) Система додає нову таблицю до списку таблиць.
Розширення сценаріїв	4.1) Користувач натискає на кнопку «Cancel», закриваючи вікно без збереження змін. 1.1) Дані не проходять валідацію і виводиться відповідне повідомлення про помилку.

Таблиця 1.2 – Варіант використання UC02

Назва	Редагування таблиці
Опис	Користувач може відредагувати існуючу таблицю
Передумови	Існує збережена таблиця
Постумови	Відредаговано таблицю
Основний сценарій	1) Користувач натискає на кнопку «Edit». 2) Відкривається вікно редагування таблиці. 3) Користувач задає нову інформацію про таблицю. 4) Користувач натискає на кнопку «Save». 5) Система валідує введені дані. 6) Система зберігає зміни.

Продовження таблиці 1.2

Розширення сценаріїв	4.1) Користувач натискає на кнопку «Cancel», закриваючи вікно без збереження змін. 1.1) Дані не проходять валідацію і виводиться відповідне повідомлення про помилку.

Таблиця 1.3 – Варіант використання UC03

Назва	Видалення таблиці
Опис	Користувач може видалити існуючу таблицю
Передумови	Існує збережена таблиця
Постумови	Видалено таблицю
Основний сценарій	1) Користувач натискає на кнопку «Delete». 2) Відображається запит на підтвердження виконання дії. 3) Користувач підтверджує виконання дії. 4) Система видаляє таблицю.
Розширення сценаріїв	3.1) Користувач відміняє виконання дії.

Таблиця 1.4 – Варіант використання UC04

Назва	Імпортування таблиць із файлу
Опис	Користувач може імпортувати таблиці із раніше створеного JSON-файлу
Передумови	Існує JSON-файл із даними в форматі, який розпізнає система
Постумови	Додано нові таблицю

Продовження таблиці 1.4

Основний сценарій	1) Користувач натискає на кнопку «Import». 2) Відкривається вікно імпортування таблиць. 3) Користувач обирає файл збережений на диску. 4) Користувач обирає режим обробки дублікатів серед даних, що імпортуються. 5) Користувач натискає на кнопку «Save». 6) Система валідує дані, що знаходяться в завантаженому файлі. 7) Система додає нові таблиці до списку таблиць.
Розширення сценаріїв	5.1) Користувач натискає на кнопку «Cancel», закриваючи вікно без збереження змін. 1.1) Дані не проходять валідацію і виводиться відповідне повідомлення про помилку.

Таблиця 1.5 – Варіант використання UC05

Назва	Експорт збережених таблиць у файл у форматі JSON
Опис	Користувач може завантажити файл із збереженими таблицями, аби їх можна було імпортувати повторно
Передумови	–
Постумови	На комп’ютер користувача завантажено JSON файл із даними про збережені таблиці
Основний сценарій	1) Користувач натискає на кнопку «Export». 2) Система серіалізує інформацію про таблиці. 3) Починається завантаження JSON-файлу із згенерованими даними.

Розширення сценаріїв	–

Таблиця 1.6 – Варіант використання UC06

Назва	Побудова запиту
Опис	Користувач може побудувати запит до бази даних
Передумови	–
Постумови	Форма побудови запиту містить інформацію, про операцію що має бути виконана
Основний сценарій	1) Користувач натискає на одну з кнопок «SELECT», «MULTISELECT», «INSERT», «UPDATE», «DELETE». 2) Система заповнює вікно побудови запиту формовими елементами, котрі відповідають обраному типу операції. 3) Користувач заповнює форму у відповідності із тим, що має відбутися, під час роботи запиту. 4) Користувач натискає на кнопку «View SQL». 5) Система відображає вікно із згенерованим SQL кодом.
Розширення сценаріїв	–

Таблиця 1.7 – Варіант використання UC07

Назва	Валідація згенерованого SQL коду
Опис	Користувач може перевірити згенерований SQL код на валідність
Передумови	Форма побудови запиту містить інформацію, про операцію що має бути виконана
Постумови	Система відображає інформацію про валідність згенерованого коду

Продовження таблиці 1.7

Основний сценарій	1) Користувач натискає на кнопку «View SQL». 2) Система відображає вікно із згенерованим SQL кодом. 3) Користувач натискає на кнопку «Validate». 4) Система перевіряє код на валідність. 5) Всередині вікна із згенерованим SQL кодом відображається повідомлення із інформацією про валідність коду або повідомлення про помилку.
Розширення сценаріїв	1.1) У разі неочікуваної відповіді від сервера, відображається відповідне повідомлення.

Таблиця 1.8 – Варіант використання UC08

Назва	Копіювання згенерованого SQL коду
Опис	Користувач може скопіювати згенерований SQL код
Передумови	Форма побудови запиту містить інформацію, про операцію що має бути виконана
Постумови	Буфер обміну містить згенерований SQL код
Основний сценарій	1) Користувач натискає на кнопку «View SQL». 2) Система відображає вікно із згенерованим SQL кодом. 3) Користувач натискає на кнопку «Copy to clipboard». 4) Згенерований SQL код копіюється в буфер обміну комп’ютера користувача.
Розширення сценаріїв	1.1) У разі помилки операції копіювання коду, відображається відповідне повідомлення.

1.4.2 Розроблення нефункціональних вимог

Розроблене програмне забезпечення повинно відповідати наступним нефункціональним вимогам:

– текст інтерфейсу користувача має бути наданий англійською мовою;

– графічний інтерфейс і функціонал реалізований на стороні клієнта має підтримуватися більшістю сучасних браузерів.

1.5 Постановка задачі

Призначенням розробки є створення програмного забезпечення побудови запитів до баз даних із конвертацією їх в програмний код написаний мовою SQL для СУБД PostgreSQL.

Метою даної роботи є надання розробникам-початківцям можливості спростити процес вивчення мови SQL шляхом використання ПЗ, котре б допомагало уникати помилок при написанні запитів і мінімізувало необхідність писати код власноруч; надання професійним розробникам інструментів генерації SQL коду із простішим інтерфейсом користувача порівняно із тим, що використовується у більшості існуючих продуктів.

Для досягнення поставлених цілей необхідно вирішити наступні задачі:

– розробити модуль роботи із абстрактними таблицями БД;

– розробити модуль побудови запитів до реляційних баз даних;

– реалізувати функціонал із автоматичної генерації програмного коду мовою SQL.

Для досягнення мети даної роботи, розроблене застосування має підтримувати наступний функціонал:

– створення таблиць;

– редагування таблиць;

– видалення таблиць;

– імпортування таблиць із файлу;

– експорт таблиць у файл;
– створення запитів;
– генерація програмного коду мовою SQL;
– надання користувачу згенерованого програмного коду.

Програмний продукт має бути виконаний у формі веб-застосування. Бізнес-логіка, не рахуючи валідації згенерованого коду, має бути реалізована на стороні клієнта, аби користувач мав змогу завантажити веб-сторінку та мати доступ до більшої частини функціоналу продукту без необхідності залежати від сервера, на якому ця сторінка зберігається.

Висновки до розділу

Існує немала кількість програмного забезпечення, створеного із ціллю допомогти людям, котрі працюють із базами даних, із написанням запитів до них. Але через те що аудиторія таких рішень сильно відрізняється рівнем знань мови SQL та використовує в роботі дуже різний її функціонал, ці рішення завжди мають специфічні риси, котрі роблять їх незручними для використання у загальному випадку. Якщо фахівці, котрі лише працюють із даними, мають великий вибір доступних програмних продуктів, що може задовольнити майже будь-кого, розробники програмного забезпечення часто змушені стикатися із продуктами, які містять проблеми або в плані доступного функціоналу, або в плані зручності та простоти використання.

У цьому розділі було наведено загальні положення та опис даної предметної області. Розглянуто різні підходи до проектування та розробки схожого програмного забезпечення та конкретні приклади існуючих програмних продуктів. Було сформульовано список функціональних та нефункціональних вимог до розроблюваного застосунку. Сформульовано призначення, мету та перелік задач, які мають бути вирішені.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання та аналіз програмного забезпечення

Перед тим як починати розробку застосування, потрібно змоделювати бізнес-процеси, які відбуватимуться під час його роботи. На рисунку документа КПІ.ІП-8135.045300.06.99СС зображено схему структурну варіанті використання – усього передбачено вісім основних операцій, які може виконувати користувач:

- створення нових таблиць;
- редагування існуючих таблиць;
- імпортування нових таблиць із файлу;
- експортування існуючих таблиць у файл;
- видалення існуючих таблиць;
- побудова запитів до бази даних;
- валідація коду побудованого запиту;
- копіювання коду побудованого запиту.

Із вищезгаданої діаграми можна побачити дві особливості системи. По-перше, операції створення таблиць, редагування таблиць та імпортування таблиць включають в себе внутрішню операцію валідації стану таблиці – іншими словами, валідації вхідних даних – котра автоматично виконується системою і не є доступною користувачу. По-друге, існує операція генерації SQL коду, яка автоматично виконується системою під час побудови запиту, яку користувач може ініціювати самостійно але тільки в рамках процесу побудови запиту, і від якої залежать операції валідації та копіювання коду, оскільки вони використовують згенерований код у якості своїх вхідних даних. Діаграми діяльності для кожного варіанту використання зображені на рисунках 2.1-2.8.

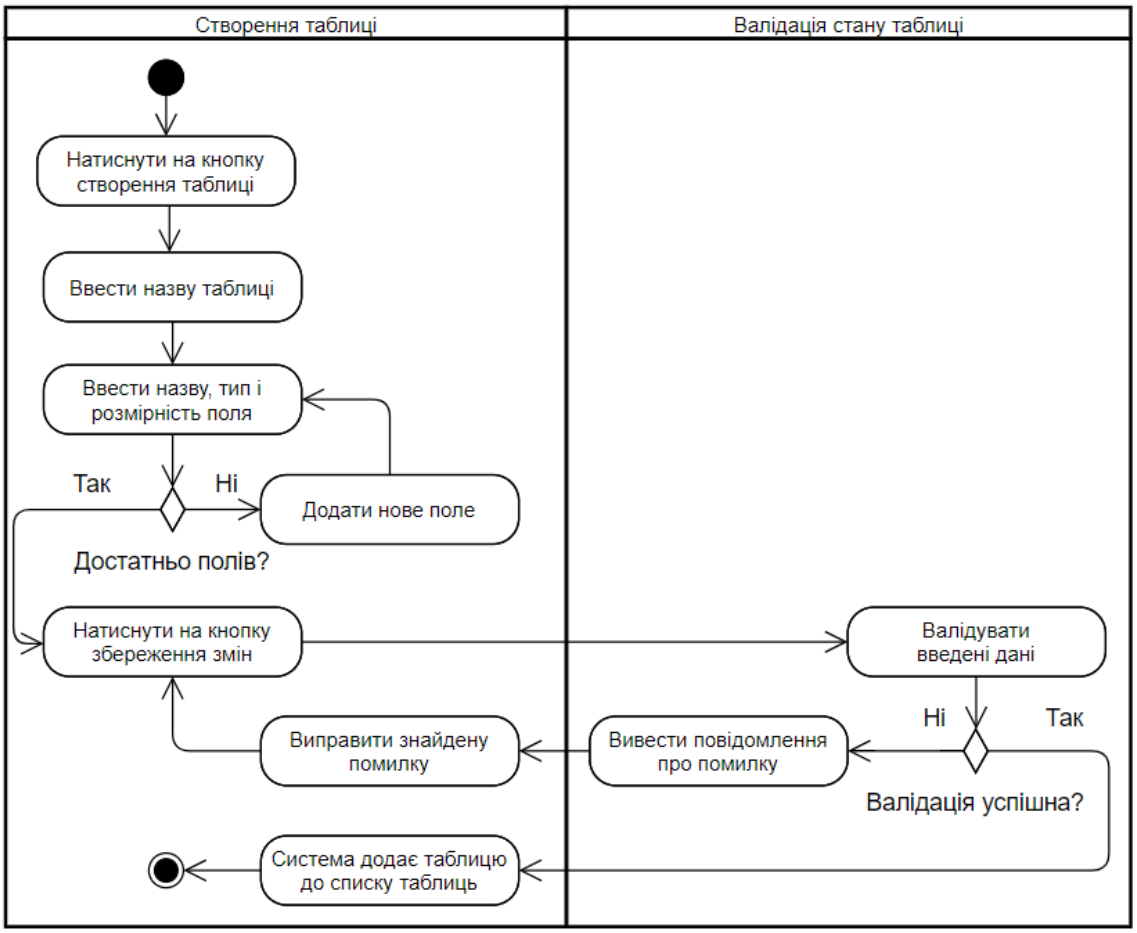


Рисунок 2.1 – Діаграма діяльності процесу створення таблиці

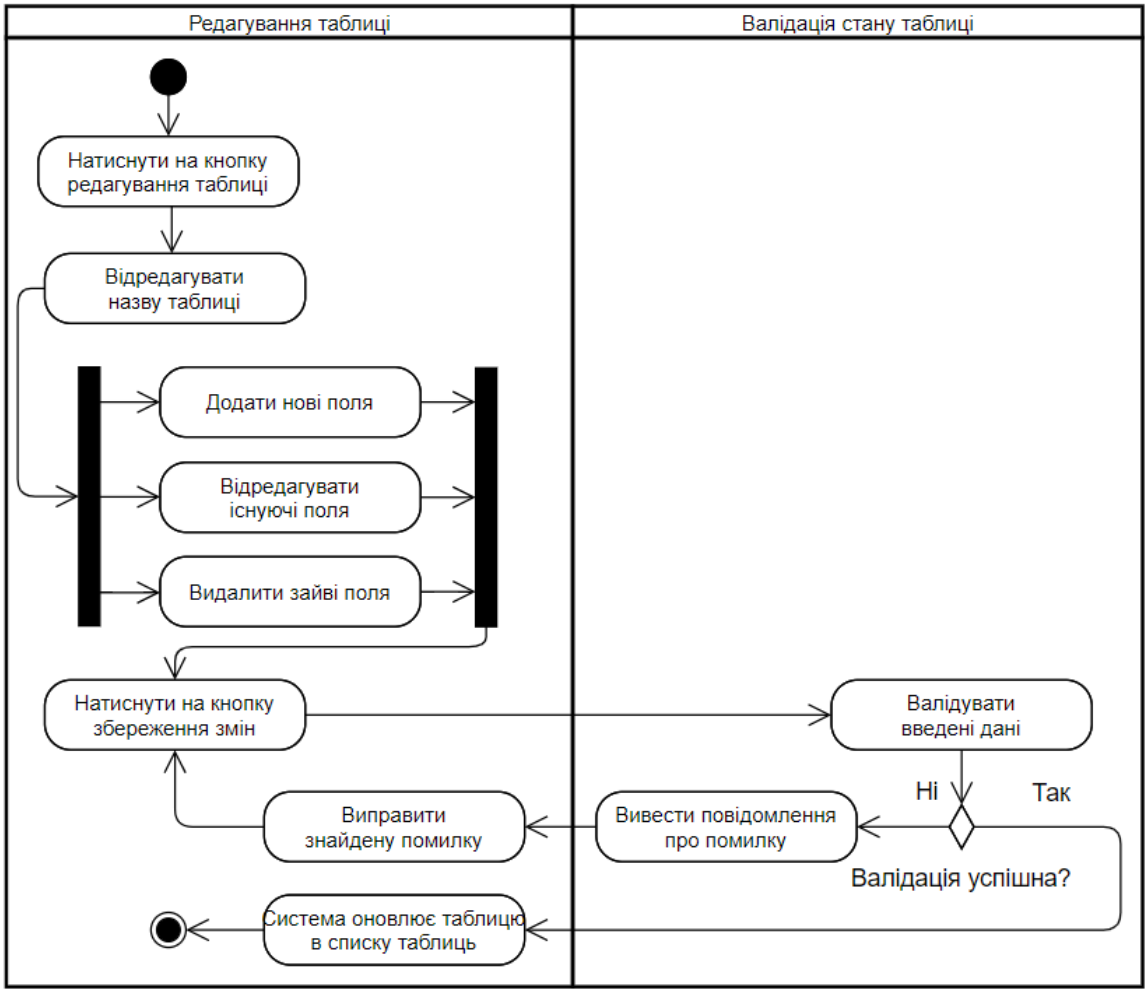


Рисунок 2.2 – Діаграма діяльності процесу редагування таблиці

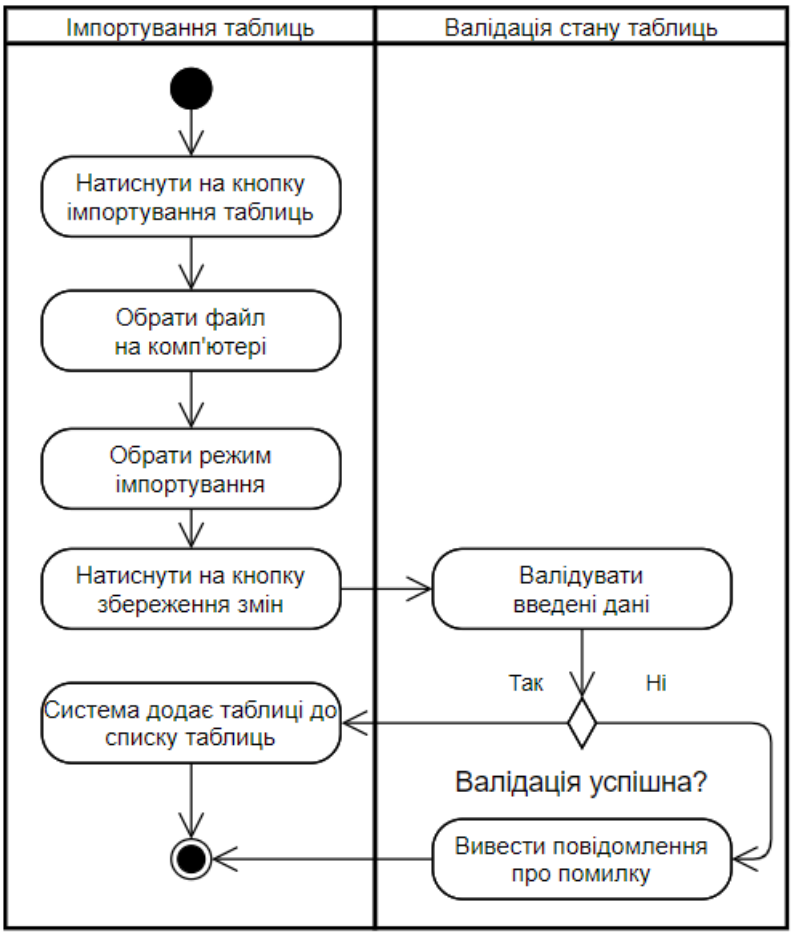


Рисунок 2.3 – Діаграма діяльності процесу імпортування таблиць

Рисунок 2.4 – Діаграма діяльності процесу експортування таблиць

Рисунок 2.5 – Діаграма діяльності процесу видалення таблиці

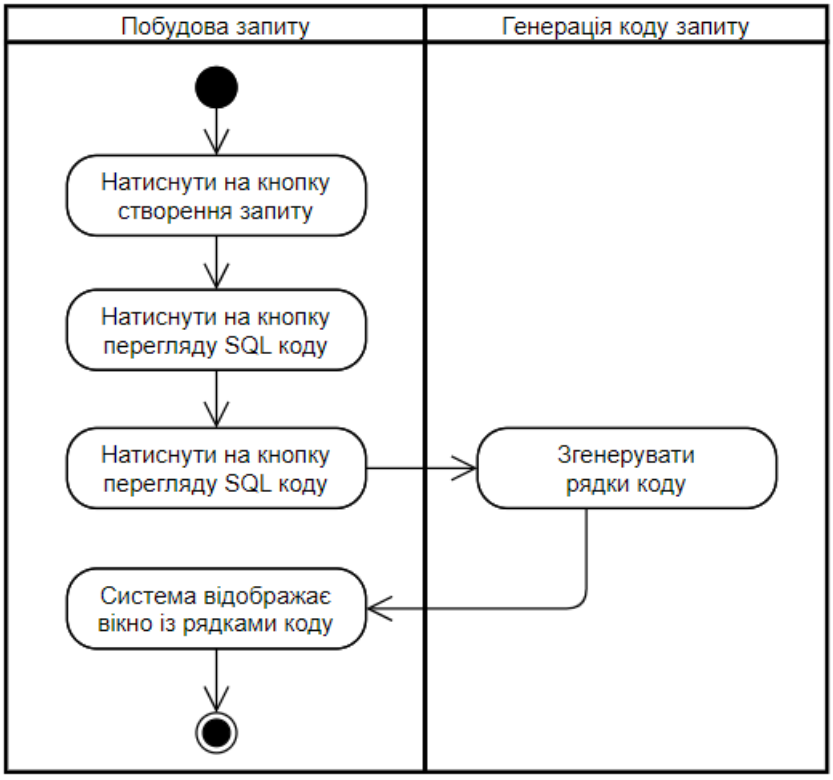


Рисунок 2.6 – Діаграма діяльності процесу побудови запиту

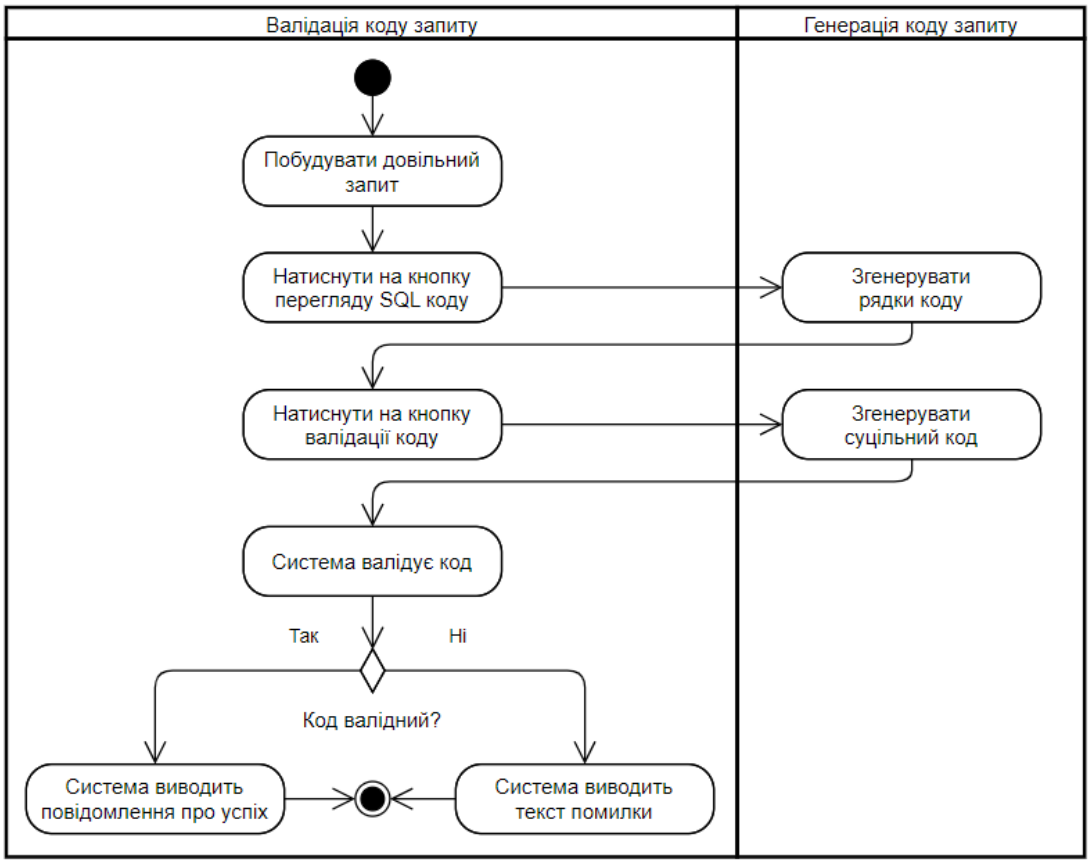


Рисунок 2.7 – Діаграма діяльності процесу валідації коду

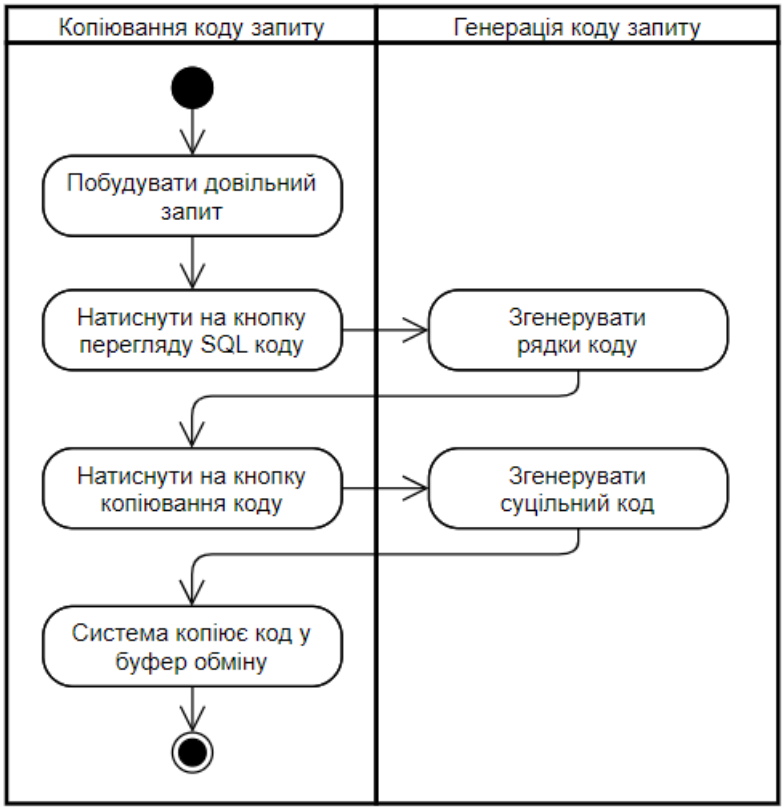


Рисунок 2.8 – Діаграма діяльності процесу копіювання коду

2.2 Архітектура програмного забезпечення

Розроблюване застосування складається з трьох основних компонентів:

– клієнта, із яким взаємодіє користувач;
– сервера, із яким клієнт обмінюється даними;
– бази даних, яка використовується сервером, для валідації згенерованого SQL коду.

Клієнт відповідає за відображення графічного інтерфейсу, який складається з інтерфейсу керування таблицями, інтерфейсу побудови запитів і вікна відображення згенерованого SQL коду. Це вікно використовує генератор SQL коду, який також залежить від вищезгаданих інтерфейсів.

Сервер відповідає за завантаження клієнтського застосування на девайс користувача і валідацію згенерованого SQL коду. Валідація відбувається за рахунок здійснення запиту до PostgreSQL бази даних.

Схема структурна компонентів даного застосування зображена на рисунку документа КПІ.ІП-8135.045300.06.99СС.

2.3 Конструювання програмного забезпечення

Серверна частина застосування – консольна програма, тексти програмного коду якої написані мовою Python із використанням бібліотеки Flask. Клієнтська частина застосування – веб-сторінка, тексти програмного коду якої написані мовами: HTML – для задання початкового її вмісту, CSS – для задання стилізації, JavaScript – для задання бізнес-логіки. Увесь код написано в імперативному стилі, подекуди, із використанням глобальних змінних, які відповідають константам або об’єктам, котрі містять контекст чи будь-які налаштування програми.

Інформація про реалізовані функції наведена в таблицях 2.1-2.2; оскільки багато з них виконують суто допоміжну роль, лише інформація про найважливіші функції наведена у вищезгаданих таблицях.

Таблиця 2.1 – Головні функції серверної частини застосунку

Назва функції	Вхідні дані	Вихідні дані	Призначення
`validate`	Рядок, рядок	Відповідь від сервера	Валідація згенерованого SQL коду
`has_semicolon`	Рядок	Булеве значення	Перевірка коду на наявність символа «;»

Таблиця 2.2 – Головні функції клієнтської частини застосунку

Назва функції	Вхідні дані	Вихідні дані	Призначення
`generateOptions`	Масив рядків	Масив HTML елементів	Генерація вмісту для HTML селекторів
`fillAddTableModal`	рядок, масив об’єктів	–	Заповнення даними форми створення таблиці
`validateIdenfitier`	Рядок	Булеве значення	Валідація SQL ідентифікаторів
`modalToTableObject`	–	Об’єкт	Створення об’єкта таблиці із форми
`validateTableData`	Об’єкт, булеве значення	Булеве значення	Валідація вмісту таблиці

Продовження таблиці 2.2

validateTableStructure	Об’єкт	Булеве значення	Валідація структури таблиці
serializeTables	–	Рядок	Серіалізація таблиць
tableObjectToTableRows	Об’єкт	Масив HTML елементів	Створення HTML рядків на основі колонок таблиці
addTableBlock	Об’єкт	–	Додавання елемента до списку таблиць
generateSqlLines	Об’єкт	Масив рядків	Генерація рядків SQL коду побудованого запиту
saveObjectChanges	–	–	Оновлення стану об’єкта підзапиту
fillViewSqlModal	–	–	Оновлення інтерфейсу перегляду SQL коду
generateSqlTables	–	Рядок	Генерація SQL коду створення таблиць

Продовження таблиці 2.2

addSelect	HTML елемент, рядок, рядок	–	Додавання SELECT-елемента в інтерфейсі побудови запиту
addGroupBy	HTML елемент, рядок	–	Додавання GROUPBY-елемента в інтерфейсі побудови запиту
addInsertField	HTML елемент, рядок	–	Додавання INSERT-елемента в інтерфейсі побудови запиту
addOrderBy	HTML елемент, рядок, рядок	–	Додавання ORDERBY-елемента в інтерфейсі побудови запиту

Продовження таблиці 2.2

`addWith`	HTML елемент, об’єкт, булеве значення, рядок	–	Додавання WITH-елемента в інтерфейсі побудови запиту
`addFrom`	HTML елемент, рядок, рядок або об’єкт, рядок, рядок, булеве значення, рядок	–	Додавання FROM-елемента в інтерфейсі побудови запиту
`addSet`	HTML елемент, рядок, список об’єктів, рядок або об’єкт	–	Додавання SET-елемента в інтерфейсі побудови запиту

Продовження таблиці 2.2

`addSubSelect`	HTML елемент, рядок або об’єкт, рядок	–	Додавання SUBSELECT-елемента в інтерфейсі побудови запиту
`fillQueryBuilderWithTopObject`	–	–	Заповнення інтерфейсу побудови запиту для актуального підзапиту
`fillQueryBuilderWithSelect`	Об’єкт	–	Заповнення інтерфейсу побудови запиту для операції SELECT
`fillQueryBuilderWithMultiselect`	Об’єкт	–	Заповнення інтерфейсу побудови запиту для операції MULTISELECT

Продовження таблиці 2.2

`fillQueryBuilderWithValues`	Об’єкт	–	Заповнення інтерфейсу побудови запиту для операції VALUES
`fillQueryBuilderWithInsert`	Об’єкт	–	Заповнення інтерфейсу побудови запиту для операції INSERT
`fillQueryBuilderWithUpdate`	Об’єкт	–	Заповнення інтерфейсу побудови запиту для операції UPDATE
`fillQueryBuilderWithDelete`	Об’єкт	–	Заповнення інтерфейсу побудови запиту для операції DELETE

Продовження таблиці 2.2

`createRootObject`	Функція	–	Ініціалізація стану підсистеми побудови запиту
`capitalize`	Рядок	Рядок	Повертає рядок записаним маленькими буквами починаючи з великої букви

2.4 Аналіз безпеки даних

Розроблюване застосування складається із трьох частин: клієнта, сервера та бази даних.

Майже весь функціонал застосування реалізований на стороні клієнту, що мінімізує необхідність обмінюватися даними із сервером. У випадку, коли обмін все ж відбувається, сервер не зберігає жодної отриманої від користувача інформації, тому можливість витоку даних відсутня. Єдина присутня вразливість у цьому місці – безпека самого процесу обміну даними через мережу Інтернет. Для того щоб нівелювати цю проблему, достатньо використовувати протокол клієнт-серверної взаємодії HTTPS, котрий забезпечує шифрування та автентифікацію даних, за рахунок використання іншого криптографічно-безпечного протоколу TLS.

Інше вразлива точка – взаємодія сервера із базою даних. Розроблюване застосування за своїм дизайном здатне приймати від користувача довільний програмний код написаний мовою SQL та виконувати його в контексті

робочої бази даних. Такий підхід може бути надзвичайно небезпечним, якщо користувач бази даних, від імені якого виконуються запити, має необмежені права. Тому, для забезпечення коректної і в той же час безпечної роботи даного функціоналу, створюється максимально обмежене середовище:

– на сервері існуватиме лише одна база даних, котра й буде використовуватися для виконання запитів користувачів застосування;

– ця база даних залишатиметься пустою – без жодних таблиць, розрізів або функцій;

– увесь SQL код буде знаходитися всередині транзакцій, котрі не будуть реально виконувати отриманий від користувача запит і завжди відкочуватимуть будь-які можливі зміни;

– усі запити будуть виконуватися від імені користувача бази даних, котрий має мінімальний необхідний набір прав – створення таблиць та їх використання;

– SQL код додатково перевіряється на наявність символа «;», який виконує роль розділювача виразів у мові SQL, у будь-якій позиції окрім останньої, аби переконатися, що користувач не намагається вплинути на вміст реального запиту до бази даних, надсилаючи кілька програмних виразів одночасно.

Висновки до розділу

У даному розділі було розглянуто бізнес-процеси розроблюваного застосування, описано та змодельовано їх із використанням мови UML. Було виділено основні архітектурні елементи системи та показано, як вони взаємодіють між собою і з яких компонентів вони складаються. Надано перелік ключових функцій у програмній реалізації застосування разом із інформацією про їх вхідні дані, вихідні дані та призначення. Було проаналізовано, які вразливі місця існують з погляду безпеки, та запропоновано методи їх захисту.

3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз якості ПЗ

Ціллю даного плану являється опис процесу тестування веб-застосування для побудови запитів до бази даних із використанням графічного інтерфейсу.

У процесі виконання роботи було створено план розробки ПЗ, визначено функціональні та не функціональні вимоги, реалізовано основний функціонал кожної з підсистем. Наступним кроком розробки застосування є перевірка його на коректність у роботі та, відповідно, готовність до використання.

Усі елементи, з яких складається система, мають бути протестовані. Основні складові елементи розроблюваного ПЗ перераховано в таблиці 3.1.

Таблиця 3.1 – Основні складові елементи розроблюваного застосування

Елемент	Вимоги до елемента
Інтерфейс сайту	Розташовано на клієнті
Функціонал сайту	Розташовано на клієнті
Функціонал сервера	Розташовано на сервері
База даних	Розташовано на сервері

3.2 Опис процесів тестування

Майже весь функціонал системи реалізовано на клієнті, а його виконання досягається шляхом взаємодії із формовими елементами веб-сторінки. Більша частина бізнес-логіки є однотипною і зводиться до створення правильних формових елементів, заповнення їх даними та зчитування даних із них для різних складових мови SQL. Окремі складові застосунку не мають впливу на коректність роботи один одного, з чого слідує

висновок, що можна швидко та легко тестувати новий функціонал у ручному режимі.

3.2.1 Тестування функціоналу клієнту

Увесь функціонал, що має бути протестований, можна розділити на наступні категорії:

– робота із таблицями – створення, редагування та видалення;

– робота із файлами – імпортування з девайсу користувача та експорт на девайс користувача;

– робота із запитами – генерація формових елементів, заповнення їх даними, зчитування даних, навігація між редакторами підзапитів;

– робота із SQL кодом – генерація коду, валідація коду.

3.2.2 Тестування інтерфейсу користувача

Інтерфейс користувача динамічно змінюється в результаті взаємодії із його елементами: кнопками, спадними списками, текстовими блоками. Оскільки генерація SQL коду та навігація між підзапитами напряму залежить від того, які елементи містить інтерфейс побудови запиту у певний момент часу, особливо важливо переконатися, що при взаємодії із ними виконуються правильні дії:

– новостворені елементи мають правильний тип і розташування;

– після пересування або видалення існуючих елементів, їхні сусіди продовжують працювати без змін у своїй логіці;

– при переході між підзапитами, нинішні дані зберігаються, а інтерфейс побудови запиту заповнюється новими релевантними даними.

3.2.3 Тестування безпеки

Оскільки валідація SQL коду виконується шляхом надсилання запиту до робочої бази даних, котра є частиною розроблюваної системи, потрібно

переконатися, що користувач застосунку не має прав для виконання будь-яких операцій, окрім створення та використання таблиць.

3.3 Опис контрольного прикладу

У процесі тестування системи було перевірено увесь її функціонал, включаючи кожен передбачений варіант використання, та кожен елемент інтерфейсу побудови запитів. Опис виконаних тестів надано в таблицях 3.2-3.28. Оскільки деякі тести повторюються для різних елементів інтерфейсу побудови запитів, можуть бути виконані із різними типами запитів або можуть бути виконані для різних вхідних даних, нижче наведено лише приблизний опис процедури тестування такого функціоналу – це стосується таблиць 3.14, 3.15, 3.16, 3.17, 3.19, 3.22, 3.25, 3.27, 3.28.

Таблиця 3.2 – Тестування процесу створення таблиці

Мета тесту	Перевірка того, що користувач може створювати таблиці
Початковий стан	Відкрита головна сторінка системи
Схема проведення тесту	1) Натиснути кнопку створення таблиці; 2) ввести назву таблиці; 3) натиснути кнопку додавання поля; 4) ввести назви полів, обрати типи полів у спадному списку, вказати значення розмірності полів; 5) натиснути кнопку збереження змін.

Продовження таблиці 3.2

Очікуваний результат	У списку таблиць таблиць відображається новий запис, де назва таблиці а також кількість, назви, типи та розмірності полів відповідають введеним значенням

Таблиця 3.3 – Тестування процесу редагування таблиці

Мета тесту	Перевірка того, що користувач може редагувати таблиці
Початковий стан	Відкрита головна сторінка системи, існує принаймні одна таблиця із, як мінімум, двома полями
Схема проведення тесту	1) Натиснути кнопку редагування під одним із записів у списку таблиць; 2) змінити назву таблиці; 3) видалити одне поле таблиці; 4) змінити існуючі значення для другого поля таблиці; 5) додати нове поле таблиці; 6) натиснути кнопку збереження змін.
Очікуваний результат	Запис, який відповідає таблиці, що редагувалася, містить нову назву таблицю, відредаговане поле із новими даними і нове поле

Таблиця 3.4 – Тестування процесу видалення таблиці

Мета тесту	Перевірка того, що користувач може видаляти таблиці
Початковий стан	Відкрита головна сторінка системи, існує принаймні одна таблиця

Продовження таблиці 3.4

Схема проведення тесту	1) Натиснути кнопку видалення під одним із записів у списку таблиць; 2) натиснути кнопку підтвердження у спливаючому вікні.
Очікуваний результат	Запис, який відповідає таблиці, що видалялася, відсутній

Таблиця 3.5 – Тестування процесу експортування таблиць

Мета тесту	Перевірка того, що користувач може експортувати таблиці у форматі JSON
Початковий стан	Відкрита головна сторінка системи, існує кілька таблиць
Схема проведення тесту	Натиснути кнопку експортування.
Очікуваний результат	На девайсі користувача почнеться завантаження файлу у форматі JSON, котрий містить інформацію про всі існуючі таблиці

Таблиця 3.6 – Тестування процесу імпортування таблиць

Мета тесту	Перевірка того, що користувач може імпортувати таблиці із JSON файлу за відсутності конфліктів
Початковий стан	Відкрита головна сторінка системи
Схема проведення тесту	1) Створити довільну таблицю; 2) натиснути кнопку експортування; 3) видалити попередньо створену таблицю; 4) натиснути кнопку імпортування; 5) обрати попередньо завантажений файл у полі вибору файлу з пам’яті девайсу; 6) натиснути кнопку збереження змін.

Продовження таблиці 3.6

Очікуваний результат	У списку таблиць з’явиться запис, зміст якого відповідає експортованим даним

Таблиця 3.7 – Тестування процесу імпортування таблиць у режимі ігнорування дублікатів

Мета тесту	Перевірка того, що користувач може імпортувати таблиці із JSON файлу ігноруючи дані, котрі конфліктують із таблицями збереженими в системі
Початковий стан	Відкрита головна сторінка системи
Схема проведення тесту	1) Створити довільну таблицю; 2) натиснути кнопку експортування; 3) відредагувати попередньо створену таблицю; 4) натиснути кнопку імпортування; 5) обрати попередньо завантажений файл у полі вибору файлу з пам’яті девайсу; 6) обрати режим ігнорування дублікатів; 7) натиснути кнопку збереження змін.
Очікуваний результат	Запис у списку таблиць, створений і відредагований протягом виконання тесту, не змінився

Таблиця 3.8 – Тестування процесу імпортування таблиць у режимі переписування дублікатів

Мета тесту	Перевірка того, що користувач може імпортувати таблиці із JSON файлу переписуючи дані, котрі конфліктують із таблицями збереженими в системі
Початковий стан	Відкрита головна сторінка системи

Продовження таблиці 3.8

Схема проведення тесту	1) Створити довільну таблицю; 2) натиснути кнопку експортування; 3) відредагувати попередньо створену таблицю; 4) натиснути кнопку імпортування; 5) обрати попередньо завантажений файл у полі вибору файлу з пам’яті девайсу; 6) обрати режим переписування дублікатів; 7) натиснути кнопку збереження змін.
Очікуваний результат	Запис у списку таблиць, створений і відредагований протягом виконання тесту, містить дані із імпортованого файлу

Таблиця 3.9 – Тестування можливості створення SELECT запиту

Мета тесту	Перевірка того, що користувач може створити SELECT запит до БД
Початковий стан	Відкрита головна сторінка системи
Схема проведення тесту	Натиснути на кнопку «SELECT».
Очікуваний результат	Інтерфейс побудови запиту містить наступні розділи: With із кнопками, для додавання WITH-елементів; Select із спадним списком типу вибірки та кнопкою додавання SELECT-елементів; From із кнопками, для додавання FROM-елементів; Where із полем для вводу виразу; Group by із спадним список режиму групування та кнопкою додавання GROUPBY-елементів; Having із полем, для вводу виразу; Order by із кнопкою для додавання ORDERBY-елементів; Limit із полем для вводу числа; Offset із полем для вводу числа

Таблиця 3.10 – Тестування можливості створення MULTISELECT запиту

Мета тесту	Перевірка того, що користувач може створити MULTISELECT запит до БД
Початковий стан	Відкрита головна сторінка системи
Схема проведення тесту	Натиснути на кнопку «MULTISELECT».
Очікуваний результат	Інтерфейс побудови запиту містить наступні розділи: With із кнопками, для додавання WITH-елементів; Subselect із кнопками, для додавання SUBSELECT-елементів; Order by із кнопкою для додавання ORDERBY-елементів; Limit із полем для вводу числа; Offset із полем для вводу числа

Таблиця 3.11 – Тестування можливості створення INSERT запиту

Мета тесту	Перевірка того, що користувач може створити INSERT запит до БД
Початковий стан	Відкрита головна сторінка системи
Схема проведення тесту	Натиснути на кнопку «INSERT».
Очікуваний результат	Інтерфейс побудови запиту містить наступні розділи: With із кнопками, для додавання WITH-елементів; Insert into із полями, для вводу назви та псевдоніма таблиці, кнопкою для додавання набору колонок таблиці; Data із спадним списком джерела даних; On conflict із спадним списком режиму вирішення конфлікту; Returning із кнопкою, для додавання RETURNING-елементів

Таблиця 3.12 – Тестування можливості створення UPDATE запиту

Мета тесту	Перевірка того, що користувач може створити UPDATE запит до БД
Початковий стан	Відкрита головна сторінка системи
Схема проведення тесту	Натиснути на кнопку «UPDATE».
Очікуваний результат	Інтерфейс побудови запиту містить наступні розділи: With із кнопками, для додавання WITH-елементів; Update із полями, для вводу назви та псевдоніма таблиці; Set із кнопками, для додавання наборів колонок та джерел даних; From із кнопками, для додавання FROM-елементів; Where із полем, для вводу виразу; Returning із кнопкою, для додавання RETURNING-елементів

Таблиця 3.13 – Тестування можливості створення DELETE запиту

Мета тесту	Перевірка того, що користувач може створити DELETE запит до БД
Початковий стан	Відкрита головна сторінка системи
Схема проведення тесту	Натиснути на кнопку «DELETE».
Очікуваний результат	Інтерфейс побудови запиту містить наступні розділи: With із кнопками, для додавання WITH-елементів; Delete from із полями, для вводу назви та псевдоніма таблиці; Using із кнопками, для додавання USING-елементів; Where із полем, для вводу виразу; Returning із кнопкою, для додавання RETURNING-елементів

Таблиця 3.14 – Приклад тестування процесу навігації між підзапитами (для операції INSERT як WITH-елемента операції SELECT)

Мета тесту	Перевірка того, що користувач може переходити між підзапитами
Початковий стан	Відкрита головна сторінка системи
Схема проведення тесту	1) Натиснути на кнопку «SELECT»; 2) натиснути на кнопку «+insert» у розділі With; 3) натиснути на блок у рамці; 4) заповнити форму довільними даними; 5) натиснути на кнопку «Return».
Очікуваний результат	Інтерфейс побудови запиту містить форму для операції SELECT із єдиним елементом – блоком із SQL кодом для операції INSERT

Таблиця 3.15 – Приклад тестування процесу додавання WITH-елементів до запиту

Мета тесту	Перевірка того, що користувач може додавати WITH-елементи в запиті
Початковий стан	Відкрита головна сторінка системи
Схема проведення тесту	1) Натиснути на кнопку «SELECT»; 2) натиснути на кожну кнопку в розділі With.
Очікуваний результат	Розділ With інтерфейсу побудови запиту містить 6 елементів, кожен із яких складається із прапорця «Recursive», поля імені, блока підзапиту та контрольних кнопок

Таблиця 3.16 – Приклад тестування процесу додавання SELECT-елементів до запиту

Мета тесту	Перевірка того, що користувач може додавати SELECT-елементи в запиті
Початковий стан	Відкрита головна сторінка системи
Схема проведення тесту	1) Натиснути на кнопку «SELECT»; 2) кілька разів натиснути на кнопку «+» у розділі Select.
Очікуваний результат	Розділ Select інтерфейсу побудови запиту містить кілька елементів, що складаються із поля виразу, поля псевдоніма та контрольних кнопок

Таблиця 3.17 – Приклад тестування процесу додавання FROM-елементів до запиту

Мета тесту	Перевірка того, що користувач може додавати FROM-елементи в запиті
Початковий стан	Відкрита головна сторінка системи
Схема проведення тесту	1) Натиснути на кнопку «SELECT»; 2) натиснути на кожну кнопку в розділі From.
Очікуваний результат	Розділ From інтерфейсу побудови запиту містить 7 елементів: перший містить поля для вводу виразу та псевдоніму; третій, п’ятий та сьомий – блок підзапиту і поле для псевдоніму; другий, четвертий та шостий – спадне меню із типом об’єднання, прапорцем «Lateral» та полем вводу виразу; поля під непарними номерами також містять контрольні кнопки

Таблиця 3.18 – Тестування процесу додавання GROUPBY-елементів до запиту

Мета тесту	Перевірка того, що користувач може додавати GROUPBY-елементи в запиті
Початковий стан	Відкрита головна сторінка системи
Схема проведення тесту	1) Натиснути на кнопку «SELECT»; 2) кілька разів натиснути на кнопку «+» в розділі Group by.
Очікуваний результат	Розділ Group by інтерфейсу побудови запиту містить кілька елементів, що складаються із поля виразу та контрольних кнопок

Таблиця 3.19 – Приклад тестування процесу додавання ORDERBY-елементів до запиту

Мета тесту	Перевірка того, що користувач може додавати ORDERBY-елементи в запиті
Початковий стан	Відкрита головна сторінка системи
Схема проведення тесту	1) Натиснути на кнопку «SELECT»; 2) кілька разів натиснути на кнопку «+» в розділі Order by.
Очікуваний результат	Розділ Order by інтерфейсу побудови запиту містить кілька елементів, що складаються із поля виразу, спадного меню напрямку сортування та контрольних кнопок

Таблиця 3.20 – Тестування процесу додавання SUBSELECT-елементів до MULTISELECT запиту

Мета тесту	Перевірка того, що користувач може додавати SUBSELECT-елементи в MULTISELECT запиті
Початковий стан	Відкрита головна сторінка системи
Схема проведення тесту	1) Натиснути на кнопку «MULTISELECT»; 2) натиснути на кожну кнопку в розділі Subselect.
Очікуваний результат	Розділ Subselect інтерфейсу побудови запиту містить 5 елементів: перший, третій і п’ятий складаються із блоку підзапиту та контрольних кнопок; другий і четвертий – спадні списки типу операції реляційної алгебри

Таблиця 3.21 – Тестування процесу конкретизації колонок таблиці для INSERT запиту

Мета тесту	Перевірка того, що користувач може конкретизувати колонки в INSERT запиті
Початковий стан	Відкрита головна сторінка системи
Схема проведення тесту	1) Натиснути на кнопку «INSERT»; 2) кілька разів натиснути на кнопку «+» в розділі Insert into.
Очікуваний результат	Розділ Insert into інтерфейсу побудови запиту містить кілька додаткових елементів, кожен із яких складається з поля для вводу назви колонки таблиці та контрольних кнопок

Таблиця 3.22 – Приклад тестування процесу вибору джерела даних для операції INSERT

Мета тесту	Перевірка того, що користувач може обрати джерело даних в INSERT запиті
Початковий стан	Відкрита головна сторінка системи
Схема проведення тесту	1) Натиснути на кнопку «INSERT»; 2) обрати будь-який елемент окрім «DEFAULT VALUES» у спадному списку в розділі Data.
Очікуваний результат	Розділ Data інтерфейсу побудови запиту містить додатковий блок підзапиту, тип якого залежить від обраного в спадному списку типу джерела даних

Таблиця 3.23 – Тестування процесу вибору оновлення таблиці в якості стратегії вирішення конфлікту при виконанні операції INSERT

Мета тесту	Перевірка того, що користувач може оновлювати поля таблиці при виникненні конфліктів під час виконання INSERT запиту
Початковий стан	Відкрита головна сторінка системи
Схема проведення тесту	1) Натиснути на кнопку «INSERT»; 2) обрати «DO UPDATE» у спадному списку в розділі On conflict; 3) натиснути на кожну з кнопок, що з’явилися в розділі On conflict; 4) натиснути кілька разів на кожну «+ field» кнопку, що з’явилася.

Продовження таблиці 3.23

Очікуваний результат	Розділ On conflict інтерфейсу побудови запиту містить: кнопки, для додавання списків колонок із джерелами даних; поля для вводу цільової таблиці та виразу; списки колонок та джерел даних. Кожен такий список містить: кнопку додавання колонки; кнопку видалення списку; блок підзапиту, якщо джерело даних не є набором виразів. Кожен елемент списку містить: поле для вводу назви колонки; контрольні кнопки; поле для вводу виразу, якщо джерело даних є набором виразів

Таблиця 3.24 – Тестування процесу вибору колонок та джерел даних для операції UPDATE

Мета тесту	Перевірка того, що користувач може обирати колонки та джерела даних для UPDATE запиту
Початковий стан	Відкрита головна сторінка системи
Схема проведення тесту	1) Натиснути на кнопку «UPDATE»; 2) натиснути на кожну з кнопок в розділі Set; 3) натиснути кілька разів на кожну «+ field» кнопку, що з’явилася.
Очікуваний результат	Розділ Set інтерфейсу побудови запиту містить списки колонок із джерелами даних. Кожен такий список містить: кнопку додавання колонки; кнопку видалення списку; блок підзапиту, якщо джерело даних не є набором виразів. Кожен елемент списку містить: поле для вводу назви колонки; контрольні кнопки; поле для вводу виразу, якщо джерело даних є набором виразів

Таблиця 3.25 – Приклад тестування процесу генерації SQL коду для побудованого запиту

Мета тесту	Перевірка того, що система може згенерувати SQL код для запиту побудованого користувачем
Початковий стан	Інтерфейс побудови запиту заповнено довільними даними
Схема проведення тесту	Натиснути на кнопку «View SQL».
Очікуваний результат	Система відображає вікно із згенерованим SQL кодом

Таблиця 3.26 – Тестування процесу копіювання згенерованого SQL коду в буфер обміну на девайсі користувача

Мета тесту	Перевірка того, що система може скопіювати згенерований SQL код у буфер обміну
Початковий стан	Інтерфейс побудови запиту заповнено довільними даними
Схема проведення тесту	1) Натиснути на кнопку «View SQL»; 2) натиснути на кнопку «Copy to clipboard».
Очікуваний результат	Буфер обміну містить згенерований SQL код

Таблиця 3.27 – Приклад тестування процесу валідації коректного SQL коду

Мета тесту	Перевірка того, що система показує повідомлення про успіх внаслідок валідації коректного SQL коду
Початковий стан	Відкрита головна сторінка системи

Продовження таблиці 3.27

Схема проведення тесту	1) Створити таблицю із назвою «my_table», котра містить одне поле «my_number» типу integer нульової розмірності; 2) створити SELECT запит із одним SELECT-елементом, котрий містить вираз «my_number + 1», і одним FROM-елементом, котрий містить вираз «my_table»; 3) натиснути на кнопку «View SQL»; 4) натиснути на кнопку «Validate».
Очікуваний результат	Під заголовком вікна перегляду згенерованого SQL коду відображається повідомлення про успішну валідацію коду

Таблиця 3.28 – Приклад тестування процесу валідації некоректного SQL коду

Мета тесту	Перевірка того, що система показує повідомлення про помилку внаслідок валідації некоректного SQL коду
Початковий стан	Відкрита головна сторінка системи
Схема проведення тесту	1) Створити SELECT запит із одним SELECT-елементом, котрий містить вираз «my_number + 1», і одним FROM-елементом, котрий містить вираз «my_table»; 2) натиснути на кнопку «View SQL»; 3) натиснути на кнопку «Validate».
Очікуваний результат	Під заголовком вікна перегляду згенерованого SQL коду відображається повідомлення про помилку

Висновки до розділу

У даному розділі було описано процеси забезпечення якості розроблюваного програмного забезпечення. Було перераховано основні архітектурні компоненти, із яких складається дане застосування, і надано загальний опис характеристик ПЗ, що мають бути протестовані, разом із планом тестування для кожної наявної функції, котра є доступною користувачу.

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розгортання програмного забезпечення

Розроблене програмне забезпечення складається із трьох ключових частин: клієнтського застосування, серверного застосування та бази даних. Клієнтське застосування реалізовано у формі веб-сторінки, тому воно не потребує розгортання. Серверне застосування реалізовано у формі програми мовою Python 3, в ролі СУБД використовується PostgreSQL. Для того щоб їх розгорнути, необхідно підготувати середовище, яке підтримує обидві технології – вони обидві є кросплатформними, тому можна використати сервер із будь-якою сучасною операційною системою на базі Linux або Unix. Вимог щодо версії мови програмування Python 3 немає, але версія СУБД PostgreSQL має бути не менше 9.3, оскільки та є найпершою, котра підтримує всі конструкції мови SQL, які використовує розроблене застосування при генерації програмного коду.

Процес розгортання системи є наступним:

1) встановити середовище виконання Python 3 та СУБД PostgreSQL на сервері;
2) створити окрему базу даних;
3) створити користувача баз даних, котрий має доступ до обмеженого набору функцій SQL у рамках вищезгаданої БД;
4) запустити серверне застосування.

4.2 Робота з програмним забезпеченням

Серверне застосування, після свого запуску, продовжить працювати самостійно. Подальша його робота не потребує втручання розробників або системних адміністраторів, не рахуючи вирішення позаштатних або аварійних ситуацій, котрі можуть виникнути із системою на програмному чи

апаратному рівні. Детальна інструкція по роботі з програмним забезпеченням наведена в документі «Керівництво користувача».

Висновки до розділу

У даному розділі було описано вимоги до середовища, у якому може працювати розроблене програмне забезпечення, надано інструкцію із розгортання застосування та інформацію, про його подальшу підтримку.

ВИСНОВКИ

У рамках даного дипломного проєкту було спроектовано та розроблено веб-застосування для побудови запитів до баз даних PostgreSQL із використанням графічного інтерфейсу. Воно складається із клієнтського застосування, із яким взаємодіє користувач, і серверного застосування разом із базою даних, які використовуються системою під час роботи. Вибір розробити дане застосування у формі веб-додатку був зроблений з причини кросплатформності – єдине, що потрібно користувачу, аби ним скористатися, це мати встановлений браузер на своєму ПК. Серверне застосування було створено із використанням мови Python через її доступність та простоту використання. У якості цільової СУБД було обрано PostgreSQL, тому що вона є однією із найбільш використовуваних на сьогоднішній день.

Розроблене застосування в першу чергу має значущість у сфері освіти, оскільки воно надає досить широкий набір функціоналу, при цьому залишаючись простим у використанні. Хоча воно являється досить примітивним порівняно із існуючими аналогами, котрі були спеціально створені для професійних розробників програмного забезпечення, воно також може знайти використання і в цій сфері, оскільки покриває ключовий функціонал мови SQL, котрий використовується найчастіше.

Тема створення допоміжних інструментів для розробки програмного забезпечення не є суттєво важливою з наукової точки зору і має значущість лише в специфічних сферах їх повсякденного використання. Звідси можна зробити висновок, що аналогічні роботу та дослідження слід проводити лише спираючись на потреби реальних користувачів.

В розділі «Аналіз вимог до програмного забезпечення» було розглянуто існуюче аналогічне програмне забезпечення разом із їх перевагами та недоліками, наведені можливі варіанти її використання та визначено перелік функціональних та нефункціональних вимог.

На основі отриманих результатів в розділі «Моделювання та конструювання програмного забезпечення» наведено опис архітектури програмного забезпечення, обґрунтування засобів розробки, наведено специфікацію методів передбаченого та реалізованого в ньому функціоналу.

Задля впевненості в його якості та коректності в розділі «Аналіз якості та тестування програмного забезпечення» наведено план тестування, який передбачає перевірку всіх доступних користувачу функцій, всіх складових системи та загальної її безпеки, після чого той був використаний для тестування застосування на контрольному прикладі.

В розділі «Впровадження та супровід програмного забезпечення» наведено методику розгортання та роботи з розробленим програмним забезпеченням.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Alan Beaulieu Learning SQL, 3rd Edition. Sebastopol, 2020

2) Alex Martelli, Anna Ravenscroft, Steve Holden Python in a nutshell, 3rd Edition. Sebastopol, 2015

3) Miguel Grinberg Flask Web Development, 2nd Edition. Sebastopol, 2018

4) Regina O. Obe, Leo S. Hsu PostgreSQL: Up and Running, 3rd Edition. Sebastopol, 2017

5) Thread: Running untrusted sql safely?. URL: https://postgrespro.com/list/thread-id/1481054

6) Types of Software Testing: Different Testing Types with Details. URL: https://www.softwaretestinghelp.com/types-of-software-testing/

7) UML Diagram Types Guide: Learn About All Types of UML Diagrams with Examples. URL: https://creately.com/blog/diagrams/uml-diagram-types-examples/

8) Документація СУБД PostgreSQL 10. URL: https://www.postgresql.org/docs/10/index.html

9) Документація мови Python 3.9. URL: https://docs.python.org/3.9/

10) Документація фреймворку Flask. URL: https://flask.palletsprojects.com/en/2.1.x/

ДОДАТОК А. ЗВІТ ПРО ПОДІБНІСТЬ

UNICHECK

Имя пользователя:
Лісовиченко Олег Іванович

ID проверки:
1011480872

Дата проверки:
07.06.2022 07:36:32 EEST

Тип проверки:
Doc vs Internet + Library

Дата отчета:
07.06.2022 07:43:58 EEST

ID пользователя:
76913

Название файла: **ІП-81_Федоров_ПЗ**

Количество страниц: **48** Количество слов: **7484** Количество символов: **56284** Размер файла: **888.32 KB** ID файла: **1011358337**

Обнаружены модификации текста (могут влиять на процент совпадений)

2.41%
Совпадения

Наибольшее совпадение: **1%** с источником из Библиотеки (ID файла: **1011349326**)

0.9% Источники из Интернета | 12 ..... Страница 50

2.41% Источники из Библиотеки | 115 ..... Страница 50

0% Цитат

Исключение цитат выключено

Исключение списка библиографических ссылок выключено

0% Исключений

Нет исключенных источников

Модификации

Обнаружены модификации текста. Подробная информация доступна в онлайн-отчете.

Замененные символы | 5

Подозрительное форматирование | 8 страниц

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2022 р.

Веб-застосування для побудови запитів до бази даних PostgreSQL із використанням графічного інтерфейсу

Опис програми

КПІ.ІП-8135.045300.03.13

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Катерина ЛІЩУК

Нормоконтроль:
_____ Катерина ЛІЩУК

Виконавець:
_____ Олександр ФЕДОРОВ

Київ – 2022

ТЕКСТ ПРОГРАМНОГО КОДУ

Тексти програмного коду

Веб-застосування для побудови запитів до бази даних PostgreSQL із використанням графічного інтерфейсу

(Найменування програми (документа))

CD-R

(Вид носія даних)

32 арк, 111 Кб

(Обсяг програми, арк., Кб)

Київ – 2022

Файл main.py

```
from flask import Flask, render_template, request
import psycopg2
import re

app = Flask(__name__)
template = """
BEGIN;
{}
{}
ROLLBACK;
""".strip("\n")


@app.route("/", methods=["GET"])
def index():
    return render_template("index.html")


@app.route("/validate", methods=["POST"])
def validate():
    data = request.json
    if has_semicolon(data["queryCode"][:-1]):
        return {"success": False, "message": "Unsafe code received"}, 400
    query = template.format(data["tablesCode"], data["queryCode"])

    conn = psycopg2.connect(dbname="query_testing", user="testing_user",
password="1234", options="-c statement timeout=5000")
    cursor = conn.cursor()

    try:
        cursor.execute(query)
    except Exception as e:
        error_offset = len(data["tablesCode"].split("\n")) + 1
        a = str(e).split("\n")
        for i, line in enumerate(a):
            if line.startswith("LINE "):
                x, y = line.split(":", 1)
                new_line = f"{x[:5]}{int(x[5:]) - error_offset}:{y}"
                a[i] = new_line
                a[i+1] = a[i+1][len(line)-len(new_line):]
        result = {
            "success": False,
            "message": "\n".join(a)
        }
    else:
        result = {"success": True, "message": "Validated successfully"}

    cursor.close()
    conn.close()

    return result


def has_semicolon(code):
    end_quote = None
    for x in re.findall(r"\$\w*\$|.", code):
        if end_quote is None and x == ";":
            return True
        if end_quote is None and (x == "'" or len(x) > 1):
            end_quote = x
```

```
        elif x == end_quote:
            end_quote = None
    return False


if __name__ == "__main__":
    app.run(host="0.0.0.0")
```

Файл index.html

```
<style>
    body, button, input[type="text"], input[type="number"],
input[type="file"], select, table {
        font-family: monospace;
        font-size: 0.9rem;
    }

    .row {
        display: flex;
        flex-wrap: wrap;
        align-items: center;
        height: 100%;
    }

    .column-1 {
        flex: 1;
        height: 100%;
        overflow: auto;
    }

    .column-2 {
        flex: 2;
        height: 100%;
        overflow: auto;
    }

    .left-border {
        border-left: 1px solid black;
    }

    .modal {
        display: none;
        position: fixed;
        z-index: 1;
        left: 0;
        top: 0;
        width: 100%;
        height: 100%;
        background-color: rgb(0, 0, 0);
        background-color: rgba(0, 0, 0, 0.4);
    }

    .modal-content {
        background-color: #fefefe;
        position: relative;
        top: 10%;
        margin: auto;
        padding: 20px;
        border: 1px solid #888;
```

```
        width: 50%;
    }

    #add-table-modal-body {
        max-height: 60%;
        overflow: auto;
    }

    #view-sql-modal-body {
        max-height: 60%;
        overflow: auto;
    }

    .confirm-button {
        border-color: #28a745;
        background-color: #28a745;
        color: white;
    }

    .reject-button {
        border-color: #dc3545;
        background-color: #dc3545;
        color: white;
    }

    select:invalid {
        color: gray;
    }

    select:invalid>option {
        color: black;
    }

    table, tr, th, td {
        border: 1px solid black;
        border-collapse: collapse;
    }

    .borderless-table, .borderless-table>tr, .borderless-table>tr>td {
        border: none;
    }

    input[type="radio"], label {
        vertical-align: middle;
    }
</style>

<div id="add-table-modal" class="modal">
    <div class="modal-content">
        <p style="display: inline">Table name:</p>
        <input type="text" placeholder="Table name" style="margin-left: 5px"
id="table-name-input">
        <button class="confirm-button" style="float: right" id="add-field-
button">Add field</button>
        <hr>
        <div id="add-table-modal-body"></div>
        <hr>
        <button id="cancel-modal-button">Cancel</button>
        <button class="confirm-button" id="save-modal-button" style="float:
right">Save</button>
    </div>
</div>
```

```
<div id="import-tables-modal" class="modal">
    <div class="modal-content">
        <h3 style="margin-top: 0; margin-bottom: 0">Import tables</h3>
        <hr>
        <input type="file" id="import-tables-file-input" accept=".json"
multiple>
        <br>
        <br>
        <input type="radio" id="duplicate-tables-ignore-button"
name="duplicate-table-resolve" value="ignore" checked>
        <label for="duplicate-tables-ignore-button">Ignore duplicate
tables</label>
        <br>
        <input type="radio" id="duplicate-tables-overwrite-button"
name="duplicate-table-resolve" value="overwrite">
        <label for="duplicate-tables-overwrite-button">Overwrite duplicate
tables</label>
        <hr>
        <button id="cancel-modal-button-2">Cancel</button>
        <button class="confirm-button" id="save-modal-button-2" style="float:
right">Save</button>
    </div>
</div>

<div id="view-sql-modal" class="modal">
    <div class="modal-content" style="min-height: 10%">
        <h3 style="display: inline">Generated SQL:</h3>
        <div id="validation-result-block" style="overflow: auto"></div>
        <hr>
        <div id="view-sql-modal-body"></div>
        <hr>
        <button id="close-modal-button">Close</button>
        <button class="confirm-button" id="validate-button" style="float:
right">Validate</button>
        <button id="copy-to-clipboard-button" style="float: right; margin-
right: 5px">Copy to clipboard</button>
    </div>
</div>

<div class="row">
    <div id="table-data" class="column-1">
        <div style="padding: 5px">
            <button id="add-table-button" class="confirm-button">Add
table</button>
            <button id="import-tables-button" class="confirm-
button">Import</button>
            <button id="export-tables-button">Export</button>
            <hr>
            <div id="table-blocks"></div>
        </div>
    </div>
    <div id="visual-blocks-data" class="column-2 left-border">
        <div style="padding: 5px">
            <button id="topmost-select-button" class="confirm-
button">SELECT</button>
            <button id="topmost-multiselect-button" class="confirm-
button">MULTISELECT</button>
            <button id="topmost-insert-button" class="confirm-
button">INSERT</button>
            <button id="topmost-update-button" class="confirm-
button">UPDATE</button>
            <button id="topmost-delete-button" class="confirm-
button">DELETE</button>
```

```
            <button id="view-sql-button">View SQL</button>
            <button id="return-button">Return</button>
            <hr>
            <div id="query-builder-container"></div>
        </div>
    </div>
</div>

<script>
    function generateOptions(list) {
        return list.map(x => {
            option = document.createElement("option");
            option.value = x;
            option.innerHTML = x;
            return option;
        });
    }


    function fillAddTableModal(name, fields) {
        let tableNameInput = document.getElementById("table-name-input");
        tableNameInput.value = name;

        let addTableModalBody = document.getElementById("add-table-modal-
body");
        addTableModalBody.innerHTML = "";
        window.app.modalFields = [];

        for (let x of fields)
            addFieldRow(addTableModalBody, x[0], x[1], x[2]);
    }


    function validateIdentifier(s) {
        return s !== "" && s === s.trim() && /^[a-z_][a-z_0-9$]*$/.test(s);
    }


    function modalToTableObject() {
        let tableNameInput = document.getElementById("table-name-input");
        return {
            name: tableNameInput.value,
            fields: window.app.modalFields.map(x => [x[0].value, x[1].value,
+x[2].value])
        };
    }


    function validateTableData(tableObject, performAlerts) {
        let currentName = window.app.currentTable === null ? "" :
window.app.currentTable.name;
        let messageHandler = performAlerts ? alert : s => null;

        if (!validateIdentifier(tableObject.name)) {
            messageHandler("Invalid table name");
            return false;
        }
        if (window.app.tables.some(x => x.name === tableObject.name && x.name
!== currentName)) {
            messageHandler("A table with such name already exists");
            return false;
        }
        if (tableObject.fields.length === 0) {
```

```
            messageHandler("A table must have at least 1 field");
            return false;
        }

        let fieldNames = tableObject.fields.map(x => x[0]);
        if (fieldNames.some(x => !validateIdentifier(x))) {
            messageHandler("Invalid field name");
            return false;
        }
        if (fieldNames.length !== new Set(fieldNames).size) {
            messageHandler("Duplicate field name");
            return false;
        }

        let fieldTypes = tableObject.fields.map(x => x[1]);
        if (fieldTypes.some(x => x === "")) {
            messageHandler("Field type missing");
            return false;
        }

        let fieldDimensions = tableObject.fields.map(x => x[2]);
        if (fieldDimensions.some(x => isNaN(x) || x < 0 || x % 1 !== 0)) {
            messageHandler("Invalid dimensions");
            return false;
        }

        return true;
    }


    function validateTableStructure(tableObject) {
        return tableObject !== null &&
            typeof tableObject === "object" &&
            Object.keys(tableObject).length === 2 &&
            "name" in tableObject &&
            typeof tableObject.name === "string" &&
            "fields" in tableObject &&
            Array.isArray(tableObject.fields) &&
            tableObject.fields.every(x => Array.isArray(x) && x.length === 3
&& typeof x[0] === "string" && window.app.types.includes(x[1]) && typeof x[2]
=== "number");
    }


    function serializeTables() {
        let tables = window.app.tables.map(x => {
            return {
                name: x.name,
                fields: x.fields,
                dimensions: x.dimensions
            };
        });
        return JSON.stringify(tables);
    }


    function tableObjectToTableRows(tableObject) {
        let nameRow = document.createElement("tr");
        let nameTh = document.createElement("th");
        nameTh.innerHTML = tableObject.name;
        nameTh.colSpan = 2;

        nameRow.append(nameTh);
```

```
        let fieldsRows = tableObject.fields.map(x => {
            let row = document.createElement("tr");
            let nameCell = document.createElement("td");
            let typeCell = document.createElement("td");
            nameCell.innerHTML = x[0];
            typeCell.innerHTML = x[1] + "[]".repeat(x[2]);
            row.append(nameCell, typeCell);
            return row;
        });

        return [nameRow, ...fieldsRows];
    }


    function addTableBlock(tableObject) {
        let tableBlocks = document.getElementById("table-blocks");

        let block = document.createElement("div");
        block.style.marginBottom = "20px";

        let table = document.createElement("table");
        table.style.width = "100%";
        table.append(...tableObjectToTableRows(tableObject));
        tableObject.table = table;

        let editButton = document.createElement("button");
        editButton.textContent = "Edit";
        editButton.style.marginTop = "5px";
        editButton.onclick = () => {
            window.app.currentTable = tableObject;
            fillAddTableModal(tableObject.name, tableObject.fields,
tableObject.dimensions);
            let addTableModal = document.getElementById("add-table-modal");
            addTableModal.style.display = "block";
        };

        let deleteButton = document.createElement("button");
        deleteButton.textContent = "Delete";
        deleteButton.style.marginTop = "5px";
        deleteButton.style.float = "right";
        deleteButton.classList.add("reject-button");
        deleteButton.onclick = () => {
            if (confirm("Confirm deletion")) {
                tableObject.table = null;
                window.app.tables = window.app.tables.filter(x => x !==
tableObject);
                block.remove();
            }
        };

        block.append(table, editButton, deleteButton);

        window.app.tables.push(tableObject);
        tableBlocks.append(block);
    }


    function generateSqlLines(sqlObject) {
        let result = [];

        if (sqlObject.type === "select") {
            if (sqlObject.with.length) {
```

```
                generateSqlLinesWith(result, sqlObject.with);
            }

            if (sqlObject.select.length) {
                result.push(`SELECT ${sqlObject.selectType}`);
                for (let i = 0; i < sqlObject.select.length; i++) {
                    if (i) result[result.length-1] += ",";
                    let x = sqlObject.select[i];
                    let expression = `  ${x.expression}`;
                    if (x.alias) expression += ` AS ${x.alias}`;
                    result.push(expression);
                }
            }

            if (sqlObject.from.length) {
                result.push("FROM");
                generateSqlLinesDataSource(result, sqlObject.from);
            }

            if (sqlObject.where) result.push(`WHERE ${sqlObject.where}`);

            if (sqlObject.groupBy.length) {
                result.push(`GROUP BY ${sqlObject.groupByType}(`);
                for (let i = 0; i < sqlObject.groupBy.length; i++) {
                    if (i) result[result.length-1] += ",";
                    let x = sqlObject.groupBy[i];
                    let expression = `  ${x.expression}`;
                    result.push(expression);
                }
                result.push(")");
            }

            if (sqlObject.having) result.push(`HAVING ${sqlObject.having}`);

            if (sqlObject.orderBy.length) {
                result.push("ORDER BY");
                for (let i = 0; i < sqlObject.orderBy.length; i++) {
                    if (i) result[result.length-1] += ",";
                    let x = sqlObject.orderBy[i];
                    let expression = `  ${x.expression} ${x.direction}`;
                    result.push(expression);
                }
            }

            if (sqlObject.limit) result.push(`LIMIT ${sqlObject.limit}`);

            if (sqlObject.offset) result.push(`OFFSET ${sqlObject.offset}`);
        } else if (sqlObject.type === "multiselect") {
            if (sqlObject.with.length) {
                generateSqlLinesWith(result, sqlObject.with);
            }

            for (let i = 0; i < sqlObject.subselect.length; i++) {
                if (i) result.push(sqlObject.subselect[i].combination);
                result.push("(");
                for (let line of
generateSqlLines(sqlObject.subselect[i].data))
                    result.push(`  ${line}`);
                result.push(")");
            }

            if (sqlObject.orderBy.length) {
                result.push("ORDER BY");
```

```
                for (let i = 0; i < sqlObject.orderBy.length; i++) {
                    if (i) result[result.length-1] += ",";
                    let x = sqlObject.orderBy[i];
                    let expression = `  ${x.expression} ${x.direction}`;
                    result.push(expression);
                }
            }

            if (sqlObject.limit) result.push(`LIMIT ${sqlObject.limit}`);

            if (sqlObject.offset) result.push(`OFFSET ${sqlObject.offset}`);
        } else if (sqlObject.type === "values") {
            if (sqlObject.values.length) {
                result.push("VALUES");
                for (let i = 0; i < sqlObject.values.length; i++) {
                    if (i) result[result.length-1] += ",";
                    result.push(`  (${sqlObject.values[i].join(", ")})`);
                }
            }
        } else if (sqlObject.type === "insert") {
            if (sqlObject.with.length) {
                generateSqlLinesWith(result, sqlObject.with);
            }

            if (sqlObject.insert) {
                result.push(`INSERT INTO ${sqlObject.insert}`);
            }

            if (sqlObject.insert) {
                if (sqlObject.insertAlias) result[result.length-1] += ` AS
${sqlObject.insertAlias}`;
            }

            if (sqlObject.insert) {
                if (sqlObject.insertFields.length) result[result.length-1] +=
` (${sqlObject.insertFields.join(", ")})`;
            }

            if (sqlObject.insert) {
                if (sqlObject.insertData.type === "DEFAULT VALUES") {
                    result.push("DEFAULT VALUES");
                } else {
                    result.push("(");
                    for (let line of
generateSqlLines(sqlObject.insertData.data))
                        result.push(`  ${line}`);
                    result.push(")");
                }
            }

            if (sqlObject.onConflict.type === "DO NOTHING") {
                result.push("ON CONFLICT DO NOTHING");
            } else if (sqlObject.onConflict.type === "DO UPDATE") {
                result.push(`ON CONFLICT
(${sqlObject.onConflict.data.target}) DO UPDATE`);
                result.push("SET");
                for (let i = 0; i < sqlObject.onConflict.data.set.length;
i++) {
                    if (i) result[result.length-1] += ",";
                    let x = sqlObject.onConflict.data.set[i];
                    if (x.type === "expression") {
                        result.push(`  (${x.fields.join(", ")}) = ROW
(${x.data.join(", ")})`);
```

```
                    } else if (x.type === "select") {
                        result.push(`  (${x.fields.join(", ")}) = (`);
                        for (let line of generateSqlLines(x.data))
                            result.push(`    ${line}`);
                        result.push("  )");
                    }
                }
                if (sqlObject.onConflict.data.where) result.push(`WHERE
${sqlObject.onConflict.data.where}`);
            }

            if (sqlObject.returning.length) {
                result.push("RETURNING");
                for (let i = 0; i < sqlObject.returning.length; i++) {
                    if (i) result[result.length-1] += ",";
                    let x = sqlObject.returning[i];
                    let expression = `  ${x.expression}`;
                    if (x.alias) expression += ` AS ${x.alias}`;
                    result.push(expression);
                }
            }
        } else if (sqlObject.type === "update") {
            if (sqlObject.with.length) {
                generateSqlLinesWith(result, sqlObject.with);
            }

            if (sqlObject.update) {
                result.push(`UPDATE ${sqlObject.update}`);
            }

            if (sqlObject.update) {
                if (sqlObject.updateAlias) result[result.length-1] += ` AS
${sqlObject.updateAlias}`;
            }

            if (sqlObject.set.length) {
                result.push("SET");
                for (let i = 0; i < sqlObject.set.length; i++) {
                    if (i) result[result.length-1] += ",";
                    let x = sqlObject.set[i];
                    if (x.type === "expression") {
                        result.push(`  (${x.fields.join(", ")}) = ROW
(${x.data.join(", ")})`);
                    } else if (x.type === "select") {
                        result.push(`  (${x.fields.join(", ")}) = (`);
                        for (let line of generateSqlLines(x.data))
                            result.push(`    ${line}`);
                        result.push("  )");
                    }
                }
            }

            if (sqlObject.from.length) {
                result.push("FROM");
                generateSqlLinesDataSource(result, sqlObject.from);
            }

            if (sqlObject.where) result.push(`WHERE ${sqlObject.where}`);

            if (sqlObject.returning.length) {
                result.push("RETURNING");
                for (let i = 0; i < sqlObject.returning.length; i++) {
                    if (i) result[result.length-1] += ",";
```

```
                    let x = sqlObject.returning[i];
                    let expression = `  ${x.expression}`;
                    if (x.alias) expression += ` AS ${x.alias}`;
                    result.push(expression);
                }
            }
        } else if (sqlObject.type === "delete") {
            if (sqlObject.with.length) {
                generateSqlLinesWith(result, sqlObject.with);
            }

            if (sqlObject.from) {
                result.push(`DELETE FROM ${sqlObject.from}`);
            }

            if (sqlObject.from) {
                if (sqlObject.fromAlias) result[result.length-1] += ` AS
${sqlObject.fromAlias}`;
            }

            if (sqlObject.using.length) {
                result.push("USING");
                generateSqlLinesDataSource(result, sqlObject.using);
            }

            if (sqlObject.where) result.push(`WHERE ${sqlObject.where}`);

            if (sqlObject.returning.length) {
                result.push("RETURNING");
                for (let i = 0; i < sqlObject.returning.length; i++) {
                    if (i) result[result.length-1] += ",";
                    let x = sqlObject.returning[i];
                    let expression = `  ${x.expression}`;
                    if (x.alias) expression += ` AS ${x.alias}`;
                    result.push(expression);
                }
            }
        }

        return result;
    }


    function saveObjectChanges() {
        let sqlObject =
window.app.queryBuilderStack[window.app.queryBuilderStack.length-1];

        if (sqlObject.type === "select") {
            let withList = document.getElementById("with-list");
            let selectTypeSelector = document.getElementById("select-type-
selector");
            let selectList = document.getElementById("select-list");
            let fromList = document.getElementById("from-list");
            let whereInput = document.getElementById("where-input");
            let groupByTypeSelector = document.getElementById("group-by-type-
selector");
            let groupByList = document.getElementById("group-by-list");
            let havingInput = document.getElementById("having-input");
            let orderByList = document.getElementById("order-by-list");
            let limitInput = document.getElementById("limit-input");
            let offsetInput = document.getElementById("offset-input");

            saveWithChanges(sqlObject, withList);
```

```
            sqlObject.selectType = selectTypeSelector.value;
            sqlObject.select = [...selectList.childNodes].map(selectRow => {
                let selectRowChildren = selectRow.childNodes;
                return {
                    expression: selectRowChildren[0].value,
                    alias: selectRowChildren[2].value
                };
            });
            saveDataSourceChanges(sqlObject.from, fromList);
            sqlObject.where = whereInput.value || null;
            sqlObject.groupByType = groupByTypeSelector.value;
            sqlObject.groupBy = [...groupByList.childNodes].map(groupByRow =>
{
                let groupByRowChildren = groupByRow.childNodes;
                return {
                    expression: groupByRowChildren[0].value
                };
            });
            sqlObject.having = havingInput.value || null;
            sqlObject.orderBy = [...orderByList.childNodes].map(orderByRow =>
{
                let orderByRowChildren = orderByRow.childNodes;
                return {
                    expression: orderByRowChildren[0].value,
                    direction: orderByRowChildren[1].value
                };
            });
            sqlObject.limit = limitInput.value || null;
            sqlObject.offset = offsetInput.value || null;
        } else if (sqlObject.type === "multiselect") {
            let withList = document.getElementById("with-list");
            let subselectList = document.getElementById("sub-select-list");
            let orderByList = document.getElementById("order-by-list");
            let limitInput = document.getElementById("limit-input");
            let offsetInput = document.getElementById("offset-input");

            saveWithChanges(sqlObject, withList);
            sqlObject.subselect = [];
            for (let i = 0; i < subselectList.childNodes.length; i += 2) {
                let subselectRowChildren =
subselectList.childNodes[i].childNodes;
                let subselectObject = {
                    data: subselectRowChildren[0].underlyingSqlObject,
                    combination: i ? subselectList.childNodes[i-
1].childNodes[0].value : "UNION ALL"
                };
                sqlObject.subselect.push(subselectObject);
            }
            sqlObject.orderBy = [...orderByList.childNodes].map(orderByRow =>
{
                let orderByRowChildren = orderByRow.childNodes;
                return {
                    expression: orderByRowChildren[0].value,
                    direction: orderByRowChildren[1].value
                };
            });
            sqlObject.limit = limitInput.value || null;
            sqlObject.offset = offsetInput.value || null;
        } else if (sqlObject.type === "values") {
            let valuesTable = document.getElementById("values-table");

            sqlObject.values = [...valuesTable.childNodes].slice(0, -
1).map(valuesRow => {
```

```
                return [...valuesRow.childNodes].slice(0, -1).map(x =>
x.childNodes[0].value);
            });
        } else if (sqlObject.type === "insert") {
            let withList = document.getElementById("with-list");
            let insertInput = document.getElementById("insert-input");
            let insertAliasInput = document.getElementById("insert-alias-
input");
            let insertFieldsList = document.getElementById("insert-fields-
list");
            let insertDataContainer = document.getElementById("insert-data-
container");
            let insertDataSelector = document.getElementById("insert-data-
selector");
            let onConflictTarget = document.getElementById("on-conflict-
target");
            let onConflictWhere = document.getElementById("on-conflict-
where");
            let onConflictSelector = document.getElementById("on-conflict-
selector");
            let setList = document.getElementById("set-list");
            let returningList = document.getElementById("returning-list");

            sqlObject.insert = insertInput.value;
            sqlObject.insertAlias = insertAliasInput.value;
            sqlObject.insertFields =
[...insertFieldsList.childNodes].map(insertFieldRow => {
                return insertFieldRow.childNodes[0].value;
            });
            sqlObject.insertData = {
                type: insertDataSelector.value,
                data: insertDataContainer.innerHTML ?
insertDataContainer.childNodes[0].underlyingSqlObject : null
            };
            sqlObject.onConflict = {
                type: onConflictSelector.value,
                data: null
            };
            if (onConflictSelector.value === "DO UPDATE") {
                sqlObject.onConflict.data = {
                    target: onConflictTarget.value,
                    set: [],
                    where: onConflictWhere.value
                };
                for (let i = 0; i < setList.childNodes.length; i++) {
                    let setBlockChildren = setList.childNodes[i].childNodes;
                    let setObject = {
                        fields: []
                    };
                    if (setBlockChildren.length === 2) {
                        setObject.type = "expression";
                        setObject.data = [];
                        let rows = setBlockChildren[1].childNodes;
                        for (let i = 0; i < rows.length; i++) {

setObject.fields.push(rows[i].childNodes[0].value);
                            setObject.data.push(rows[i].childNodes[1].value);
                        }
                    } else {
                        setObject.type = "select";
                        setObject.data =
setBlockChildren[0].underlyingSqlObject;
                        let rows = setBlockChildren[2].childNodes;
```

```
                        for (let i = 0; i < rows.length; i++)
setObject.fields.push(rows[i].childNodes[0].value);
                    }
                    sqlObject.onConflict.data.set.push(setObject);
                }
            }
            sqlObject.returning =
[...returningList.childNodes].map(returningRow => {
                let returningRowChildren = returningRow.childNodes;
                return {
                    expression: returningRowChildren[0].value,
                    alias: returningRowChildren[2].value
                };
            });
        } else if (sqlObject.type === "update") {
            let withList = document.getElementById("with-list");
            let updateInput = document.getElementById("update-input");
            let updateAliasInput = document.getElementById("update-alias-
input");
            let setList = document.getElementById("set-list");
            let fromList = document.getElementById("from-list");
            let whereInput = document.getElementById("where-input");
            let returningList = document.getElementById("returning-list");

            saveWithChanges(sqlObject, withList);
            sqlObject.update = updateInput.value;
            sqlObject.updateAlias = updateAliasInput.value;
            sqlObject.set = [];
            for (let i = 0; i < setList.childNodes.length; i++) {
                let setBlockChildren = setList.childNodes[i].childNodes;
                let setObject = {
                    fields: []
                };
                if (setBlockChildren.length === 2) {
                    setObject.type = "expression";
                    setObject.data = [];
                    let rows = setBlockChildren[1].childNodes;
                    for (let i = 0; i < rows.length; i++) {
                        setObject.fields.push(rows[i].childNodes[0].value);
                        setObject.data.push(rows[i].childNodes[1].value);
                    }
                } else {
                    setObject.type = "select";
                    setObject.data = setBlockChildren[0].underlyingSqlObject;
                    let rows = setBlockChildren[2].childNodes;
                    for (let i = 0; i < rows.length; i++)
                        setObject.fields.push(rows[i].childNodes[0].value);
                }
                sqlObject.set.push(setObject);
            }
            saveDataSourceChanges(sqlObject.from, fromList);
            sqlObject.where = whereInput.value;
            sqlObject.returning =
[...returningList.childNodes].map(returningRow => {
                let returningRowChildren = returningRow.childNodes;
                return {
                    expression: returningRowChildren[0].value,
                    alias: returningRowChildren[2].value
                };
            });
        } else if (sqlObject.type === "delete") {
            let withList = document.getElementById("with-list");
```

```
            let fromInput = document.getElementById("from-input");
            let fromAliasInput = document.getElementById("from-alias-input");
            let usingList = document.getElementById("using-list");
            let whereInput = document.getElementById("where-input");
            let returningList = document.getElementById("returning-list");

            saveWithChanges(sqlObject, withList);
            sqlObject.from = fromInput.value;
            sqlObject.fromAlias = fromAliasInput.value;
            saveDataSourceChanges(sqlObject.using, usingList);
            sqlObject.where = whereInput.value;
            sqlObject.returning =
[...returningList.childNodes].map(returningRow => {
                let returningRowChildren = returningRow.childNodes;
                return {
                    expression: returningRowChildren[0].value,
                    alias: returningRowChildren[2].value
                };
            });
        }
    }


    function fillViewSqlModal() {
        let sqlObject =
window.app.queryBuilderStack[window.app.queryBuilderStack.length-1];

        let validationResultBlock = document.getElementById("validation-
result-block");
        validationResultBlock.innerHTML = "";

        let sqlLines = generateSqlLines(sqlObject);
        let maxLineNumberWidth = (sqlLines.length + "").length;

        let codeContainer = document.createElement("pre");
        let code = document.createElement("code");
        for (let i = 0; i < sqlLines.length; i++) {
            let lineNumberSpan = document.createElement("span");
            lineNumberSpan.textContent = (i + 1 +
"").padStart(maxLineNumberWidth);
            lineNumberSpan.style.color = "grey";

            let codeLine = `  ${sqlLines[i]}${i < sqlLines.length - 1 ? "\n"
: ";"}`;

            code.append(lineNumberSpan, codeLine);
        }
        codeContainer.append(code);

        let viewSqlModalBody = document.getElementById("view-sql-modal-
body");
        viewSqlModalBody.innerHTML = "";
        viewSqlModalBody.append(codeContainer);
    }


    function generateSqlTables() {
        let tablesCodes = window.app.tables.map(table => {
            let lines = [];
            lines.push(`CREATE TABLE ${table.name} (`);
            for (let i = 0; i < table.fields.length; i++) {
                let [name, type, dimensions] = table.fields[i];
                if (i) lines[lines.length-1] += ",";
```

```
                lines.push(`  ${name} ${type}${"[]".repeat(dimensions)}`);
            }
            lines.push(");");
            return lines.join("\n");
        });
        return tablesCodes.join("\n");
    }


    function addSelect(selectList, expression, alias) {
        let selectRow = document.createElement("div");
        selectRow.style.marginTop = "5px";

        let expressionInput = document.createElement("input");
        expressionInput.type = "text";
        expressionInput.placeholder = "Expression";
        if (expression) expressionInput.value = expression;

        let asSpan = document.createElement("span");
        asSpan.textContent = "AS";
        asSpan.style.marginLeft = "5px";

        let aliasInput = document.createElement("input");
        aliasInput.type = "text";
        aliasInput.placeholder = "Alias";
        aliasInput.style.marginLeft = "5px";
        if (alias) aliasInput.value = alias;

        let moveUpButton = createMoveUpButton(() => {
            moveUpDirectNeighbors(selectList, selectRow);
        });

        let moveDownButton = createMoveDownButton(() => {
            moveDownDirectNeighbors(selectList, selectRow);
        });

        let deleteButton = createDeleteButton(() => {
            selectRow.remove();
        });

        selectRow.append(expressionInput, asSpan, aliasInput, moveUpButton,
moveDownButton, deleteButton);
        selectList.append(selectRow);
    }


    function addGroupBy(groupByList, expression) {
        let groupByRow = document.createElement("div");
        groupByRow.style.marginTop = "5px";

        let expressionInput = document.createElement("input");
        expressionInput.type = "text";
        expressionInput.placeholder = "Expression";
        if (expression) expressionInput.value = expression;

        let moveUpButton = createMoveUpButton(() => {
            moveUpDirectNeighbors(groupByList, groupByRow);
        });

        let moveDownButton = createMoveDownButton(() => {
            moveDownDirectNeighbors(groupByList, groupByRow);
        });
```

```
        let deleteButton = createDeleteButton(() => {
            groupByRow.remove();
        });

        groupByRow.append(expressionInput, moveUpButton, moveDownButton,
deleteButton);
        groupByList.append(groupByRow);
    }


    function addInsertField(insertFieldsList, field) {
        let insertFieldRow = document.createElement("div");
        insertFieldRow.style.marginTop = "5px";

        let fieldInput = document.createElement("input");
        fieldInput.type = "text";
        fieldInput.placeholder = "Field";
        if (field) fieldInput.value = field;

        let moveUpButton = createMoveUpButton(() => {
            moveUpDirectNeighbors(insertFieldsList, insertFieldRow);
        });

        let moveDownButton = createMoveDownButton(() => {
            moveDownDirectNeighbors(insertFieldsList, insertFieldRow);
        });

        let deleteButton = createDeleteButton(() => {
            insertFieldRow.remove();
        });

        insertFieldRow.append(fieldInput, moveUpButton, moveDownButton,
deleteButton);
        insertFieldsList.append(insertFieldRow);
    }


    function addOrderBy(orderByList, expression, direction) {
        let orderByRow = document.createElement("div");
        orderByRow.style.marginTop = "5px";

        let expressionInput = document.createElement("input");
        expressionInput.type = "text";
        expressionInput.placeholder = "Expression";
        if (expression) expressionInput.value = expression;

        let directionSelector = document.createElement("select");
        directionSelector.style.marginLeft = "5px";
        directionSelector.append(...generateDirectionOptions());
        directionSelector.value = direction;

        let moveUpButton = createMoveUpButton(() => {
            moveUpDirectNeighbors(orderByList, orderByRow);
        });

        let moveDownButton = createMoveDownButton(() => {
            moveDownDirectNeighbors(orderByList, orderByRow);
        });

        let deleteButton = createDeleteButton(() => {
            orderByRow.remove();
        });
```

```
        orderByRow.append(expressionInput, directionSelector, moveUpButton,
moveDownButton, deleteButton);
        orderByList.append(orderByRow);
    }


    function addWith(withList, sqlObject, recursive, alias) {
        let withRow = document.createElement("div");
        withRow.style.marginTop = "5px";

        let recursiveLabel = document.createElement("label");
        let recursiveCheckbox = document.createElement("input");
        recursiveCheckbox.type = "checkbox";
        if (recursive) recursiveCheckbox.checked = true;
        let recursiveSpan = document.createElement("span");
        recursiveSpan.textContent = "Recursive";

        let aliasInput = document.createElement("input");
        aliasInput.type = "text";
        aliasInput.placeholder = "Name";
        aliasInput.style.marginLeft = "5px";
        if (alias) aliasInput.value = alias;

        let moveUpButton = createMoveUpButton(() => {
            moveUpDirectNeighbors(withList, withRow);
        });

        let moveDownButton = createMoveDownButton(() => {
            moveDownDirectNeighbors(withList, withRow);
        });

        let deleteButton = createDeleteButton(() => {
            withRow.remove();
        });

        let sqlBlock = generateSqlBlock(sqlObject, {marginTop: "5px",
marginBottom: "20px"});

        recursiveLabel.append(recursiveCheckbox, recursiveSpan);
        withRow.append(recursiveLabel, aliasInput, moveUpButton,
moveDownButton, deleteButton, sqlBlock);
        withList.append(withRow);
    }


    function addFrom(fromList, type, data, alias, join, lateral, condition) {
        if (fromList.innerHTML) {
            let joinBlock = document.createElement("div");
            joinBlock.style.marginTop = "5px";

            let joinSelector = document.createElement("select");
            joinSelector.append(...generateJoinOptions());
            joinSelector.value = join;

            let lateralLabel = document.createElement("label");
            lateralLabel.style.marginLeft = "5px";
            let lateralCheckbox = document.createElement("input");
            lateralCheckbox.type = "checkbox";
            if (lateral) lateralCheckbox.checked = true;
            let lateralSpan = document.createElement("span");
            lateralSpan.textContent = "Lateral";

            let onSpan = document.createElement("span");
```

```
            onSpan.textContent = "ON";
            onSpan.style.marginLeft = "5px";

            let conditionInput = document.createElement("input");
            conditionInput.type = "text";
            conditionInput.style.marginLeft = "5px";
            conditionInput.placeholder = "Condition";
            if (condition) conditionInput.value = condition;

            lateralLabel.append(lateralCheckbox, lateralSpan);
            joinBlock.append(joinSelector, lateralLabel, onSpan,
conditionInput);
            fromList.append(joinBlock);
        }

        let fromRow = document.createElement("div");
        fromRow.style.marginTop = "5px";
        fromRow.style.marginBottom = "20px";

        if (type === "expression") {
            let expressionInput = document.createElement("input");
            expressionInput.type = "text";
            expressionInput.style.marginRight = "5px";
            expressionInput.placeholder = "Expression";
            if (data) expressionInput.value = data;

            fromRow.append(expressionInput);
        } else if (type === "select") {
            let sqlBlock = generateSqlBlock(data, {marginBottom: "5px"});

            fromRow.append(sqlBlock);
        }

        let asSpan = document.createElement("span");
        asSpan.textContent = "AS";

        let aliasInput = document.createElement("input");
        aliasInput.type = "text";
        aliasInput.placeholder = "Alias";
        aliasInput.style.marginLeft = "5px";
        if (alias) aliasInput.value = alias;

        let moveUpButton = createMoveUpButton(() => {
            moveUpNeighborsWithBetweenElement(fromList, fromRow);
        });

        let moveDownButton = createMoveDownButton(() => {
            moveDownNeighborsWithBetweenElement(fromList, fromRow);
        });

        let deleteButton = createDeleteButton(() => {
            let previousSibling = fromRow.previousSibling;
            let nextSibling = fromRow.nextSibling;
            if (!previousSibling) {
                if (nextSibling) nextSibling.remove();
            } else previousSibling.remove();
            fromRow.remove();
        });

        fromRow.append(asSpan, aliasInput, moveUpButton, moveDownButton,
deleteButton);
        fromList.append(fromRow);
    }
```

```
    function addSet(setList, type, fields, data) {
        let setBlock = document.createElement("div");

        if (type === "expression") {
            let expressionsHeader = document.createElement("h3");
            expressionsHeader.innerHTML = "Expressions";
            expressionsHeader.style.display = "inline-block";
            expressionsHeader.style.marginTop = "0";
            expressionsHeader.style.marginBottom = "0";

            let expressionsList = document.createElement("div");
            expressionsList.style.marginBottom = "10px";

            let buttonsContainer = document.createElement("div");
            buttonsContainer.style.marginTop = "15px";
            buttonsContainer.style.marginBottom = "5px";

            let addButton = document.createElement("button");
            addButton.textContent = "+ field";
            addButton.classList.add("confirm-button");
            addButton.style.marginLeft = "5px";
            addButton.onclick = () => {
                addSetExpression(expressionsList, "", "");
            };
            for (let i = 0; i < fields.length; i++)
addSetExpression(expressionsList, fields[i], data[i]);

            let deleteButton = createDeleteButton(() => {
                setBlock.remove();
            });

            buttonsContainer.append(expressionsHeader, addButton,
deleteButton);
            setBlock.append(buttonsContainer, expressionsList);
        } else if (type === "select") {
            let sqlBlock = generateSqlBlock(data, {marginTop: "15px",
marginBottom: "5px"});

            let fieldsList = document.createElement("div");
            fieldsList.style.marginBottom = "20px";

            let buttonsContainer = document.createElement("div");
            buttonsContainer.style.marginBottom = "5px";

            let addButton = document.createElement("button");
            addButton.textContent = "+ field";
            addButton.classList.add("confirm-button");
            addButton.onclick = () => {
                addSetField(fieldsList, "");
            };
            for (let i = 0; i < fields.length; i++) addSetField(fieldsList,
fields[i]);

            let deleteButton = createDeleteButton(() => {
                setBlock.remove();
            });

            buttonsContainer.append(addButton, deleteButton);
            setBlock.append(sqlBlock, buttonsContainer, fieldsList);
        }
```

```
        setList.append(setBlock);
    }


    function fillQueryBuilderWithSelect(selectObject) {
        let queryBuilderContainer = document.getElementById("query-builder-
container");

        let withHeader = document.createElement("h2");
        withHeader.innerHTML = "With";
        withHeader.style.display = "inline-block";
        withHeader.style.marginBottom = "5px";
        let withList = document.createElement("div");
        withList.id = "with-list";
        let [addWithSelectButton, addWithMultiselectButton,
addWithValuesButton, addWithInsertButton, addWithUpdateButton,
addWithDeleteButton] = generateAddWithButtons(withList);
        for (let with_ of selectObject.with) addWith(withList,
with_.sqlObject, with_.recursive, with_.alias);

        let selectHeader = document.createElement("h2");
        selectHeader.innerHTML = "Select";
        selectHeader.style.display = "inline-block";
        selectHeader.style.marginBottom = "5px";
        let selectTypeSelector = document.createElement("select");
        selectTypeSelector.style.marginLeft = "5px";
        selectTypeSelector.append(...generateSelectTypeOptions());
        selectTypeSelector.value = selectObject.selectType;
        selectTypeSelector.id = "select-type-selector";
        let selectList = document.createElement("div");
        selectList.id = "select-list";
        let addSelectButton = document.createElement("button");
        addSelectButton.textContent = "+";
        addSelectButton.style.marginLeft = "5px";
        addSelectButton.classList.add("confirm-button");
        addSelectButton.onclick = () => {
            addSelect(selectList, "", "");
        };
        for (let select of selectObject.select) addSelect(selectList,
select.expression, select.alias);

        let fromHeader = document.createElement("h2");
        fromHeader.innerHTML = "From";
        fromHeader.style.display = "inline-block";
        fromHeader.style.marginBottom = "5px";
        let fromList = document.createElement("div");
        fromList.id = "from-list";
        let [addFromExpressionButton, addFromSelectButton,
addFromMultiselectButton, addFromValuesButton] =
generateFromButtons(fromList);
        for (let from of selectObject.from) addFrom(fromList, from.type,
from.data, from.alias, from.join, from.lateral, from.condition);

        let whereHeader = document.createElement("h2");
        whereHeader.innerHTML = "Where";
        whereHeader.style.marginBottom = "0";
        let whereInput = document.createElement("input");
        whereInput.id = "where-input";
        whereInput.type = "text";
        whereInput.placeholder = "Condition";
        if (selectObject.where) whereInput.value = selectObject.where;

        let groupByHeader = document.createElement("h2");
```

```
        groupByHeader.innerHTML = "Group by";
        groupByHeader.style.display = "inline-block";
        groupByHeader.style.marginBottom = "5px";
        let groupByTypeSelector = document.createElement("select");
        groupByTypeSelector.style.marginLeft = "5px";
        groupByTypeSelector.append(...generateGroupByTypeOptions());
        groupByTypeSelector.value = selectObject.groupByType;
        groupByTypeSelector.id = "group-by-type-selector";
        let groupByList = document.createElement("div");
        groupByList.id = "group-by-list";
        let addGroupByButton = document.createElement("button");
        addGroupByButton.textContent = "+";
        addGroupByButton.style.marginLeft = "5px";
        addGroupByButton.classList.add("confirm-button");
        addGroupByButton.onclick = () => {
            addGroupBy(groupByList, "");
        };
        for (let groupBy of selectObject.groupBy) addGroupBy(groupByList,
groupBy.expression);

        let havingHeader = document.createElement("h2");
        havingHeader.innerHTML = "Having";
        havingHeader.style.marginBottom = "0";
        let havingInput = document.createElement("input");
        havingInput.id = "having-input";
        havingInput.type = "text";
        havingInput.placeholder = "Condition";
        if (selectObject.having) havingInput.value = selectObject.having;

        let orderByHeader = document.createElement("h2");
        orderByHeader.innerHTML = "Order by";
        orderByHeader.style.display = "inline-block";
        orderByHeader.style.marginBottom = "5px";
        let orderByList = document.createElement("div");
        orderByList.id = "order-by-list";
        let addOrderByButton = document.createElement("button");
        addOrderByButton.textContent = "+";
        addOrderByButton.style.marginLeft = "5px";
        addOrderByButton.classList.add("confirm-button");
        addOrderByButton.onclick = () => {
            addOrderBy(orderByList, "", "ASC");
        };
        for (let orderBy of selectObject.orderBy) addOrderBy(orderByList,
orderBy.expression, orderBy.direction);

        let limitHeader = document.createElement("h2");
        limitHeader.innerHTML = "Limit";
        limitHeader.style.marginBottom = "0";
        let limitInput = document.createElement("input");
        limitInput.id = "limit-input";
        limitInput.type = "number";
        limitInput.placeholder = "Limit";
        limitInput.min = 0;
        if (selectObject.limit) limitInput.value = selectObject.limit;

        let offsetHeader = document.createElement("h2");
        offsetHeader.innerHTML = "Offset";
        offsetHeader.style.marginBottom = "0";
        let offsetInput = document.createElement("input");
        offsetInput.id = "offset-input";
        offsetInput.type = "number";
        offsetInput.placeholder = "Offset";
        offsetInput.min = 0;
```

```
        if (selectObject.offset) offsetInput.value = selectObject.offset;

        queryBuilderContainer.innerHTML = "";
        queryBuilderContainer.append(
            withHeader, addWithSelectButton, addWithMultiselectButton,
addWithValuesButton, addWithInsertButton, addWithUpdateButton,
addWithDeleteButton, withList, document.createElement("div"),
            selectHeader, selectTypeSelector, addSelectButton, selectList,
document.createElement("div"),
            fromHeader, addFromExpressionButton, addFromSelectButton,
addFromMultiselectButton, addFromValuesButton, fromList,
document.createElement("div"),
            whereHeader, whereInput, document.createElement("div"),
            groupByHeader, groupByTypeSelector, addGroupByButton,
groupByList, document.createElement("div"),
            havingHeader, havingInput, document.createElement("div"),
            orderByHeader, addOrderByButton, orderByList,
document.createElement("div"),
            limitHeader, limitInput, document.createElement("div"),
            offsetHeader, offsetInput
        );
    }


    function addSubselect(subselectList, data, combination) {
        if (subselectList.innerHTML) {
            let combinationBlock = document.createElement("div");
            combinationBlock.style.marginTop = "5px";

            let combinationSelector = document.createElement("select");
            combinationSelector.append(...generateCombinationOptions());
            combinationSelector.value = combination;

            combinationBlock.append(combinationSelector);
            subselectList.append(combinationBlock);
        }

        let subselectRow = document.createElement("div");
        subselectRow.style.marginTop = "5px";
        subselectRow.style.marginBottom = "20px";

        let moveUpButton = createMoveUpButton(() => {
            moveUpNeighborsWithBetweenElement(subselectList, subselectRow);
        });

        let moveDownButton = createMoveDownButton(() => {
            moveDownNeighborsWithBetweenElement(subselectList, subselectRow);
        });

        let deleteButton = createDeleteButton(() => {
            let previousSibling = subselectRow.previousSibling;
            let nextSibling = subselectRow.nextSibling;
            if (!previousSibling) {
                if (nextSibling) nextSibling.remove();
            } else previousSibling.remove();
            subselectRow.remove();
        });

        let sqlBlock = generateSqlBlock(data, {marginBottom: "5px"});

        subselectRow.append(sqlBlock, moveUpButton, moveDownButton,
deleteButton);
        subselectList.append(subselectRow);
```

```
    }


    function fillQueryBuilderWithMultiselect(multiselectObject) {
        let queryBuilderContainer = document.getElementById("query-builder-
container");

        let withHeader = document.createElement("h2");
        withHeader.innerHTML = "With";
        withHeader.style.display = "inline-block";
        withHeader.style.marginBottom = "5px";
        let withList = document.createElement("div");
        withList.id = "with-list";
        let [addWithSelectButton, addWithMultiselectButton,
addWithValuesButton, addWithInsertButton, addWithUpdateButton,
addWithDeleteButton] = generateAddWithButtons(withList);
        for (let with_ of multiselectObject.with) addWith(withList,
with_.sqlObject, with_.recursive, with_.alias);

        let subselectHeader = document.createElement("h2");
        subselectHeader.innerHTML = "Subselect";
        subselectHeader.style.display = "inline-block";
        subselectHeader.style.marginBottom = "5px";
        let subselectList = document.createElement("div");
        subselectList.id = "sub-select-list";
        let addSubselectSelectButton = document.createElement("button");
        addSubselectSelectButton.textContent = "+ select";
        addSubselectSelectButton.style.marginLeft = "5px";
        addSubselectSelectButton.classList.add("confirm-button");
        addSubselectSelectButton.onclick = () => {
            let emptySelectObject = createEmptySelectObject();
            addSubselect(subselectList, emptySelectObject, "UNION ALL");
        };
        let addSubselectMultiselectButton = document.createElement("button");
        addSubselectMultiselectButton.textContent = "+ multiselect";
        addSubselectMultiselectButton.style.marginLeft = "5px";
        addSubselectMultiselectButton.classList.add("confirm-button");
        addSubselectMultiselectButton.onclick = () => {
            let emptyMultiselectObject = createEmptyMultiselectObject();
            addSubselect(subselectList, emptyMultiselectObject, "UNION ALL");
        };
        let addSubselectValuesButton = document.createElement("button");
        addSubselectValuesButton.textContent = "+ values";
        addSubselectValuesButton.style.marginLeft = "5px";
        addSubselectValuesButton.classList.add("confirm-button");
        addSubselectValuesButton.onclick = () => {
            let emptyValuesObject = createEmptyValuesObject();
            addSubselect(subselectList, emptyValuesObject, "UNION ALL");
        };
        for (let subselect of multiselectObject.subselect)
addSubselect(subselectList, subselect.data, subselect.combination);

        let orderByHeader = document.createElement("h2");
        orderByHeader.innerHTML = "Order by";
        orderByHeader.style.display = "inline-block";
        orderByHeader.style.marginBottom = "5px";
        let orderByList = document.createElement("div");
        orderByList.id = "order-by-list";
        let addOrderByButton = document.createElement("button");
        addOrderByButton.textContent = "+";
        addOrderByButton.style.marginLeft = "5px";
        addOrderByButton.classList.add("confirm-button");
        addOrderByButton.onclick = () => {
```

```
            addOrderBy(orderByList, "", "ASC");
        };
        for (let orderBy of multiselectObject.orderBy)
addOrderBy(orderByList, orderBy.expression, orderBy.direction);

        let limitHeader = document.createElement("h2");
        limitHeader.innerHTML = "Limit";
        limitHeader.style.marginBottom = "0";
        let limitInput = document.createElement("input");
        limitInput.id = "limit-input";
        limitInput.type = "number";
        limitInput.placeholder = "Limit";
        limitInput.min = 0;
        if (multiselectObject.limit) limitInput.value =
multiselectObject.limit;

        let offsetHeader = document.createElement("h2");
        offsetHeader.innerHTML = "Offset";
        offsetHeader.style.marginBottom = "0";
        let offsetInput = document.createElement("input");
        offsetInput.id = "offset-input";
        offsetInput.type = "number";
        offsetInput.placeholder = "Offset";
        offsetInput.min = 0;
        if (multiselectObject.offset) offsetInput.value =
multiselectObject.offset;

        queryBuilderContainer.innerHTML = "";
        queryBuilderContainer.append(
            withHeader, addWithSelectButton, addWithMultiselectButton,
addWithValuesButton, addWithInsertButton, addWithUpdateButton,
addWithDeleteButton, withList, document.createElement("div"),
            subselectHeader, addSubselectSelectButton,
addSubselectMultiselectButton, addSubselectValuesButton, subselectList,
document.createElement("div"),
            orderByHeader, addOrderByButton, orderByList,
document.createElement("div"),
            limitHeader, limitInput, document.createElement("div"),
            offsetHeader, offsetInput
        );
    }


    function fillQueryBuilderWithValues(valuesObject) {
        let queryBuilderContainer = document.getElementById("query-builder-
container");

        let valuesHeader = document.createElement("h2");
        valuesHeader.innerHTML = "Values";
        valuesHeader.style.display = "inline-block";
        valuesHeader.style.marginBottom = "5px";

        let valuesTable = document.createElement("table");
        valuesTable.classList.add("borderless-table");
        valuesTable.id = "values-table";
        valuesTable.columnCount = 0;
        valuesTable.rowCount = 0;

        let valuesTableLastRow = document.createElement("tr");
        let valuesTableEmptyCorner = document.createElement("td");

        valuesTableLastRow.append(valuesTableEmptyCorner);
        valuesTable.append(valuesTableLastRow);
```

```
        let addColumnButton = document.createElement("button");
        addColumnButton.textContent = "+ column";
        addColumnButton.style.marginLeft = "5px";
        addColumnButton.classList.add("confirm-button");
        addColumnButton.onclick = () => {
            addColumn(valuesTable, []);
        };

        let addRowButton = document.createElement("button");
        addRowButton.textContent = "+ row";
        addRowButton.style.marginLeft = "5px";
        addRowButton.classList.add("confirm-button");
        addRowButton.onclick = () => {
            addRow(valuesTable, []);
        };

        for (let i = 0; valuesObject.values.length && i <
valuesObject.values[0].length; i++) addColumn(valuesTable, []);
        for (let values of valuesObject.values) addRow(valuesTable, values);

        queryBuilderContainer.innerHTML = "";
        queryBuilderContainer.append(
            valuesHeader, addColumnButton, addRowButton, valuesTable
        );
    }


    function fillQueryBuilderWithInsert(insertObject) {
        let queryBuilderContainer = document.getElementById("query-builder-
container");

        let withHeader = document.createElement("h2");
        withHeader.innerHTML = "With";
        withHeader.style.display = "inline-block";
        withHeader.style.marginBottom = "5px";
        let withList = document.createElement("div");
        withList.id = "with-list";
        let [addWithSelectButton, addWithMultiselectButton,
addWithValuesButton, addWithInsertButton, addWithUpdateButton,
addWithDeleteButton] = generateAddWithButtons(withList);
        for (let with_ of insertObject.with) addWith(withList,
with_.sqlObject, with_.recursive, with_.alias);

        let insertHeader = document.createElement("h2");
        insertHeader.innerHTML = "Insert into";
        insertHeader.style.marginBottom = "5px";
        let insertInput = document.createElement("input");
        insertInput.id = "insert-input";
        insertInput.type = "text";
        insertInput.placeholder = "Table";
        if (insertObject.insert) insertInput.value = insertObject.insert;
        let asSpan = document.createElement("span");
        asSpan.textContent = "AS";
        asSpan.style.marginLeft = "5px";
        let insertAliasInput = document.createElement("input");
        insertAliasInput.id = "insert-alias-input";
        insertAliasInput.type = "text";
        insertAliasInput.placeholder = "Alias";
        insertAliasInput.style.marginLeft = "5px";
        if (insertObject.insertAlias) insertAliasInput.value =
insertObject.insertAlias;
        let insertFieldsList = document.createElement("div");
```

```
        insertFieldsList.id = "insert-fields-list";
        let addInsertFieldButton = document.createElement("button");
        addInsertFieldButton.textContent = "+";
        addInsertFieldButton.style.marginLeft = "5px";
        addInsertFieldButton.classList.add("confirm-button");
        addInsertFieldButton.onclick = () => {
            addInsertField(insertFieldsList, "");
        };
        for (let insertField of insertObject.insertFields)
addInsertField(insertFieldsList, insertField);

        let insertDataHeader = document.createElement("h2");
        insertDataHeader.innerHTML = "Data";
        insertDataHeader.style.marginBottom = "5px";
        insertDataHeader.style.display = "inline-block";
        let insertDataContainer = document.createElement("div");
        insertDataContainer.id = "insert-data-container";
        let insertDataSelector = document.createElement("select");
        insertDataSelector.append(...generateInsertDataOptions());
        insertDataSelector.id = "insert-data-selector";
        insertDataSelector.value = insertObject.insertData.type;
        insertDataSelector.style.marginLeft = "5px";
        insertDataSelector.onchange = () => {
            if (insertDataSelector.value === "DEFAULT VALUES") {
                insertDataContainer.innerHTML = "";
            } else {
                let emptyObject = {
                    "VALUES": createEmptyValuesObject,
                    "SELECT": createEmptySelectObject,
                    "MULTISELECT": createEmptyMultiselectObject
                }[insertDataSelector.value]();
                setInsertData(insertDataContainer, emptyObject);
            }
        };
        if (insertObject.insertData.type !== "DEFAULT VALUES")
setInsertData(insertDataContainer, insertObject.insertData.data);

        let onConflictHeader = document.createElement("h2");
        onConflictHeader.innerHTML = "On conflict";
        onConflictHeader.style.marginBottom = "5px";
        onConflictHeader.style.display = "inline-block";
        let onConflictContainer = document.createElement("div");
        onConflictContainer.style.display = "none";
        let setList = document.createElement("div");
        setList.id = "set-list";
        let buttonsList = document.createElement("div");
        buttonsList.style.display = "none";
        buttonsList.style.marginTop = "5px";
        let [addSetExpressionsButton, addSetSelectButton,
addSetMultiselectButton, addSetValuesButton] = generateSetButtons(setList);
        buttonsList.append(addSetExpressionsButton, addSetSelectButton,
addSetMultiselectButton, addSetValuesButton);
        let onConflictTarget = document.createElement("input");
        onConflictTarget.type = "text";
        onConflictTarget.placeholder = "Target";
        onConflictTarget.id = "on-conflict-target";
        let onConflictWhere = document.createElement("input");
        onConflictWhere.type = "text";
        onConflictWhere.placeholder = "Condition";
        onConflictWhere.id = "on-conflict-where";
        onConflictWhere.style.marginLeft = "5px";
        onConflictContainer.append(onConflictTarget, onConflictWhere,
setList);
```

```
        let onConflictSelector = document.createElement("select");
        onConflictSelector.append(...generateOnConflictOptions());
        onConflictSelector.id = "on-conflict-selector";
        onConflictSelector.value = insertObject.onConflict.type;
        onConflictSelector.style.marginLeft = "5px";
        onConflictSelector.onchange = () => {
            onConflictTarget.value = "";
            onConflictWhere.value = "";
            setList.innerHTML = "";
            if (onConflictSelector.value === "DO UPDATE") {
                buttonsList.style.display = "inline-block";
                onConflictContainer.style.display = "block";
            } else {
                buttonsList.style.display = "none";
                onConflictContainer.style.display = "none";
            }
        };
        if (insertObject.onConflict.type === "DO UPDATE") {
            buttonsList.style.display = "inline-block";
            onConflictContainer.style.display = "block";
            onConflictTarget.value = insertObject.onConflict.data.target;
            onConflictWhere.value = insertObject.onConflict.data.where;
            for (let set of insertObject.onConflict.data.set) addSet(setList,
set.type, set.fields, set.data);
        }

        let returningHeader = document.createElement("h2");
        returningHeader.innerHTML = "Returning";
        returningHeader.style.display = "inline-block";
        returningHeader.style.marginBottom = "5px";
        let returningList = document.createElement("div");
        returningList.id = "returning-list";
        let addReturningButton = document.createElement("button");
        addReturningButton.textContent = "+";
        addReturningButton.style.marginLeft = "5px";
        addReturningButton.classList.add("confirm-button");
        addReturningButton.onclick = () => {
            addSelect(returningList, "", "");
        };
        for (let returning of insertObject.returning)
addSelect(returningList, returning.expression, returning.alias);

        queryBuilderContainer.innerHTML = "";
        queryBuilderContainer.append(
            withHeader, addWithSelectButton, addWithMultiselectButton,
addWithValuesButton, addWithInsertButton, addWithUpdateButton,
addWithDeleteButton, withList, document.createElement("div"),
            insertHeader, insertInput, asSpan, insertAliasInput,
addInsertFieldButton, insertFieldsList, document.createElement("div"),
            insertDataHeader, insertDataSelector, insertDataContainer,
document.createElement("div"),
            onConflictHeader, onConflictSelector, buttonsList,
onConflictContainer, document.createElement("div"),
            returningHeader, addReturningButton, returningList
        );
    }


    function fillQueryBuilderWithUpdate(updateObject) {
        let queryBuilderContainer = document.getElementById("query-builder-
container");

        let withHeader = document.createElement("h2");
```

```
        withHeader.innerHTML = "With";
        withHeader.style.display = "inline-block";
        withHeader.style.marginBottom = "5px";
        let withList = document.createElement("div");
        withList.id = "with-list";
        let [addWithSelectButton, addWithMultiselectButton,
addWithValuesButton, addWithInsertButton, addWithUpdateButton,
addWithDeleteButton] = generateAddWithButtons(withList);
        for (let with_ of updateObject.with) addWith(withList,
with_.sqlObject, with_.recursive, with_.alias);

        let updateHeader = document.createElement("h2");
        updateHeader.innerHTML = "Update";
        updateHeader.style.marginBottom = "0";
        let updateInput = document.createElement("input");
        updateInput.id = "update-input";
        updateInput.type = "text";
        updateInput.placeholder = "Table";
        if (updateObject.update) updateInput.value = updateObject.update;
        let asSpan = document.createElement("span");
        asSpan.textContent = "AS";
        asSpan.style.marginLeft = "5px";
        let updateAliasInput = document.createElement("input");
        updateAliasInput.id = "update-alias-input";
        updateAliasInput.type = "text";
        updateAliasInput.placeholder = "Alias";
        updateAliasInput.style.marginLeft = "5px";
        if (updateObject.updateAlias) updateAliasInput.value =
updateObject.updateAlias;

        let setHeader = document.createElement("h2");
        setHeader.innerHTML = "Set";
        setHeader.style.display = "inline-block";
        setHeader.style.marginBottom = "5px";
        let setList = document.createElement("div");
        setList.id = "set-list";
        let [addSetExpressionsButton, addSetSelectButton,
addSetMultiselectButton, addSetValuesButton] = generateSetButtons(setList);
        for (let set of updateObject.set) addSet(setList, set.type,
set.fields, set.data);

        let fromHeader = document.createElement("h2");
        fromHeader.innerHTML = "From";
        fromHeader.style.display = "inline-block";
        fromHeader.style.marginBottom = "5px";
        let fromList = document.createElement("div");
        fromList.id = "from-list";
        let [addFromExpressionButton, addFromSelectButton,
addFromMultiselectButton, addFromValuesButton] =
generateFromButtons(fromList);
        for (let from of updateObject.from) addFrom(fromList, from.type,
from.data, from.alias, from.join, from.lateral, from.condition);

        let whereHeader = document.createElement("h2");
        whereHeader.innerHTML = "Where";
        whereHeader.style.marginBottom = "0";
        let whereInput = document.createElement("input");
        whereInput.id = "where-input";
        whereInput.type = "text";
        whereInput.placeholder = "Condition";
        if (updateObject.where) whereInput.value = updateObject.where;

        let returningHeader = document.createElement("h2");
```

```
        returningHeader.innerHTML = "Returning";
        returningHeader.style.display = "inline-block";
        returningHeader.style.marginBottom = "5px";
        let returningList = document.createElement("div");
        returningList.id = "returning-list";
        let addReturningButton = document.createElement("button");
        addReturningButton.textContent = "+";
        addReturningButton.style.marginLeft = "5px";
        addReturningButton.classList.add("confirm-button");
        addReturningButton.onclick = () => {
            addSelect(returningList, "", "");
        };
        for (let returning of updateObject.returning)
addSelect(returningList, returning.expression, returning.alias);

        queryBuilderContainer.innerHTML = "";
        queryBuilderContainer.append(
            withHeader, addWithSelectButton, addWithMultiselectButton,
addWithValuesButton, addWithInsertButton, addWithUpdateButton,
addWithDeleteButton, withList, document.createElement("div"),
            updateHeader, updateInput, asSpan, updateAliasInput,
document.createElement("div"),
            setHeader, addSetExpressionsButton, addSetSelectButton,
addSetMultiselectButton, addSetValuesButton, setList,
document.createElement("div"),
            fromHeader, addFromExpressionButton, addFromSelectButton,
addFromMultiselectButton, addFromValuesButton, fromList,
document.createElement("div"),
            whereHeader, whereInput, document.createElement("div"),
            returningHeader, addReturningButton, returningList
        );
    }


    function fillQueryBuilderWithDelete(deleteObject) {
        let queryBuilderContainer = document.getElementById("query-builder-
container");

        let withHeader = document.createElement("h2");
        withHeader.innerHTML = "With";
        withHeader.style.display = "inline-block";
        withHeader.style.marginBottom = "5px";
        let withList = document.createElement("div");
        withList.id = "with-list";
        let [addWithSelectButton, addWithMultiselectButton,
addWithValuesButton, addWithInsertButton, addWithUpdateButton,
addWithDeleteButton] = generateAddWithButtons(withList);
        for (let with_ of deleteObject.with) addWith(withList,
with_.sqlObject, with_.recursive, with_.alias);

        let fromHeader = document.createElement("h2");
        fromHeader.innerHTML = "Delete from";
        fromHeader.style.marginBottom = "0";
        let fromInput = document.createElement("input");
        fromInput.id = "from-input";
        fromInput.type = "text";
        fromInput.placeholder = "Table";
        if (deleteObject.from) fromInput.value = deleteObject.from;
        let asSpan = document.createElement("span");
        asSpan.textContent = "AS";
        asSpan.style.marginLeft = "5px";
        let fromAliasInput = document.createElement("input");
        fromAliasInput.id = "from-alias-input";
```

```
        fromAliasInput.type = "text";
        fromAliasInput.placeholder = "Alias";
        fromAliasInput.style.marginLeft = "5px";
        if (deleteObject.fromAlias) fromAliasInput.value =
deleteObject.fromAlias;

        let usingHeader = document.createElement("h2");
        usingHeader.innerHTML = "Using";
        usingHeader.style.display = "inline-block";
        usingHeader.style.marginBottom = "5px";
        let usingList = document.createElement("div");
        usingList.id = "using-list";
        let [addUsingExpressionButton, addUsingSelectButton,
addUsingMultiselectButton, addUsingValuesButton] =
generateFromButtons(usingList);
        for (let using of deleteObject.using) addFrom(usingList, using.type,
using.data, using.alias, using.join, using.lateral, using.condition);

        let whereHeader = document.createElement("h2");
        whereHeader.innerHTML = "Where";
        whereHeader.style.marginBottom = "0";
        let whereInput = document.createElement("input");
        whereInput.id = "where-input";
        whereInput.type = "text";
        whereInput.placeholder = "Condition";
        if (deleteObject.where) whereInput.value = deleteObject.where;

        let returningHeader = document.createElement("h2");
        returningHeader.innerHTML = "Returning";
        returningHeader.style.display = "inline-block";
        returningHeader.style.marginBottom = "5px";
        let returningList = document.createElement("div");
        returningList.id = "returning-list";
        let addReturningButton = document.createElement("button");
        addReturningButton.textContent = "+";
        addReturningButton.style.marginLeft = "5px";
        addReturningButton.classList.add("confirm-button");
        addReturningButton.onclick = () => {
            addSelect(returningList, "", "");
        };
        for (let returning of deleteObject.returning)
addSelect(returningList, returning.expression, returning.alias);

        queryBuilderContainer.innerHTML = "";
        queryBuilderContainer.append(
            withHeader, addWithSelectButton, addWithMultiselectButton,
addWithValuesButton, addWithInsertButton, addWithUpdateButton,
addWithDeleteButton, withList, document.createElement("div"),
            fromHeader, fromInput, asSpan, fromAliasInput,
document.createElement("div"),
            usingHeader, addUsingExpressionButton, addUsingSelectButton,
addUsingMultiselectButton, addUsingValuesButton, usingList,
document.createElement("div"),
            whereHeader, whereInput, document.createElement("div"),
            returningHeader, addReturningButton, returningList
        );
    }


    function createRootObject(func) {
        if (window.app.queryBuilderStack.length && !confirm("The old query
will be lost. Create new query?")) return;
```

```
            let emptyObject = func();
            window.app.queryBuilderStack.length = 0;
            window.app.queryBuilderStack.push(emptyObject);
            fillQueryBuilderWithTopObject();
        }


        function capitalize(s) {
            return s.slice(0, 1).toUpperCase() + s.slice(1).toLowerCase();
        }


        setupAppData();
        setupAddTableModalBehavior();
        setupImportTablesModalBehavior();
        setupExportButtonBehavior();
        setupTopmostButtons();
        setupViewSqlModalBehavior();
    </script>
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2022 р.

Веб-застосування для побудови запитів до бази даних PostgreSQL із використанням графічного інтерфейсу
Програма та методика тестування
КПІ.ІП-8135.045300.04.51

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Катерина ЛІЩУК

Нормоконтроль:
_____ Катерина ЛІЩУК

Виконавець:
_____ Олександр ФЕДОРОВ

Київ – 2022

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є веб-застосування побудови запитів до баз даних із використанням графічного інтерфейсу. Система складається із трьох частин: браузерного клієнтського застосування, серверного застосування і PostgreSQL бази даних.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

– перевірка відповідності програмного забезпечення заявленим функціональним та нефункціональним вимогам;

– перевірка коректності роботи наявного функціоналу системи;

– перевірка зручності графічного інтерфейсу користувача;

– знаходження помилок і проблем задля їх усунення.

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються такі методи:

– димне тестування – базова перевірка на наявність найбільш явних помилок, на основі якої приймається рішення того, чи готова система до повного тестування;

– інтеграційне тестування – перевірка правильності взаємодії різних компонентів системи;

– тестування безпеки – перевірка програмного забезпечення на вразливість до атак;

– приймальне тестування – перевірка того, що програмне забезпечення відповідає заявленим до нього вимогам.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування розробленого програмного забезпечення виконується вручну, з метою знаходження недоліків як у реалізованому функціоналі системи так і в зручності її використання. Для того, щоб перевірити правильність застосування у роботі, необхідно провести наступні тестування:

– статичне тестування на відповідність нефункціональним вимогам;

– динамічне тестування на відповідність функціональним вимогам;

– тестування клієнтського застосування на працездатність у різних сучасних браузерах;

– тестування інтерфейсу користувача;

– тестування зручності використання.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2022 р.

Веб-застосування для побудови запитів до бази даних PostgreSQL із використанням графічного інтерфейсу

Керівництво користувача

КПІ.ІП-8135.045300.05.34

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Катерина ЛІЩУК

Нормоконтроль:
_____ Катерина ЛІЩУК

Виконавець:
_____ Олександр ФЕДОРОВ

Київ – 2022

ЗМІСТ

1 ЗАГАЛЬНІ ВІДОМОСТІ

«Веб-застосування для побудови запитів до бази даних PostgreSQL із використанням графічного інтерфейсу» – це система для побудови запитів до баз даних. Вона складається із трьох компонентів: клієнтського застосування, серверного застосування і бази даних. Серед цих складових, користувач напряму взаємодіє із клієнтським застосуванням, реалізованим у формі веб-додатку, який, в свою чергу, обмінюється даними із сервером. Клієнтське застосування включає в себе дві підсистеми: одна дозволяє працювати із абстрактними таблицями – створювати, редагувати, видаляти, імпортувати та експортувати їх; інша дозволяє працювати із запитами – будувати, валідувати згенерований код і копіювати його.

2 ПІДГОТОВКА ДО РОБОТИ

2.1 Системні вимоги для коректної роботи

Для успішної роботи даного застосунку необхідне виконання наступних вимог:

– наявність комп’ютера зі встановленим браузером Google Chrome версії 80 або вище, Mozilla Firefox версії 72 або вище чи Edge версії 89 або вище;

– наявність доступу до Інтернету.

2.2 Завантаження застосунку

Клієнтське застосування виконано у формі веб-додатку, тому завантаження застосунку відбувається шляхом надсилання запиту до сервера, на якому той розташований. Завантажену веб-сторінку можна зберегти на комп’ютері для майбутнього використання – увесь функціонал застосування, не рахуючи валідації згенерованого SQL коду, реалізовано на стороні клієнту, тому, якщо користувач не потребує цієї функції або не має постійного доступу до мережі Інтернет, можна обмежитися збереженою версією клієнтського застосування.

2.3 Перевірка коректної роботи

Оскільки клієнтське застосування реалізовано у формі веб-додатку, воно запускається шляхом його відкриття у браузері на комп’ютері користувача.

3 РОБОТА ІЗ ЗАСТОСУНКОМ

При запуску клієнтського додатку, користувачу відображається пустий інтерфейс застосування, зображений на рисунку 3.1. У лівій частині знаходиться частина інтерфейсу, що відповідає за роботу із таблицями – цифрою 1 позначені кнопки, для їх створення, імпортування і експортування. У правій частині знаходиться частина інтерфейсу, що відповідає за роботу із запитами – цифрою 2 позначені кнопки для створення п’яти видів запитів: SELECT, MULTISELECT (поєднання кількох SELECT-ів операціями реляційної алгебри), INSERT, UPDATE, DELETE – а також кнопки для відображення згенерованого SQL коду та повернення на один рівень вверх в конструкторі запитів.

Рисунок 3.1 – Початкова сторінка додатку

Якщо користувач натисне кнопку створення таблиці, відкриється вікно із спеціалізованою формою для цього, зображеною на рисунку 3.2. У даній формі наявні наступні елементи:

1) поле для введення назви таблиці;
2) кнопка додавання нової колонки;
3) блок редагування колонок, кожен рядок якого містить: поле вводу назви, випадаючий список типів, поле вводу розмірності та кнопку видалення колонки;
4) кнопка закриття вікна без збереження змін;
5) кнопка збереження змін.

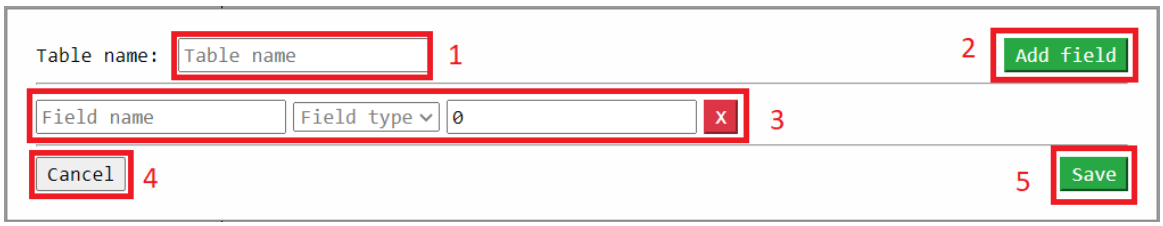


Рисунок 3.2 – Форма редагування таблиці

На рисунку 3.3 показано інтерфейс роботи із таблицями після додавання кількох таблиць. Цифрою 1 помічені кнопки редагування таблиць а цифрою 2 – кнопки видалення. Якщо натиснути на кнопку редагування, відкриється вікно показане на рисунку 3.2, заповнене існуючими даними. Якщо натиснути на кнопку видалення, відкриється стандартне вікно підтвердження дії браузера, аби користувач міг підтвердити видалення таблиці.

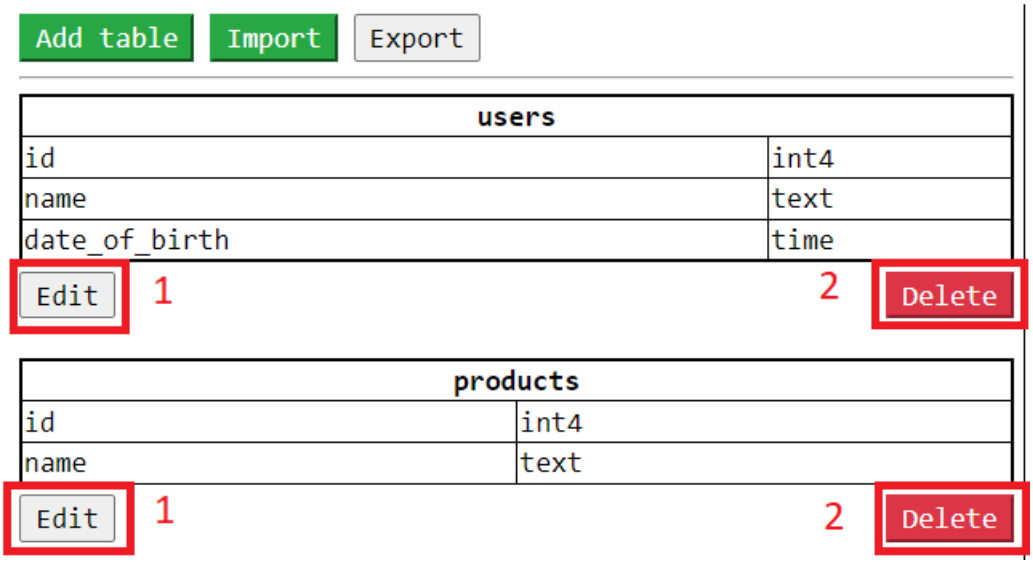


Рисунок 3.3 – Інтерфейс роботи із таблицями після додавання таблиць

Якщо користувач натисне на кнопку «SELECT», інтерфейс роботи із запитами заповниться формою для побудови SELECT-запиту, зображений на рисунку 3.4. У даній формі є:

1) кнопки додавання WITH-підзапитів;

2) випадаючий список для вибору типу операції SELECT разом із кнопкою додавання виразів;

3) кнопки додавання джерел даних;

4) поле виразу фільтрації;

5) випадаючий список типів групування разом із кнопкою додавання виразів;

6) поле виразу групової фільтрації;

7) кнопка додавання виразів сортування;

8) поле для вводу обмеження кількості повернутих рядків;

9) поле для вводу значення відступу.

Рисунок 3.4 – Інтерфейс побудови SELECT-запиту

Якщо натиснути на кнопку додавання WITH-підзапиту, нижче з'явиться відповідний блок, показаний на риснку 3.5. У даному блоці є: прапорець задання рекурсивності підзапиту, поле для вводу його назви, контрольних кнопкі та контейнер коду підзапиту. Якщо натиснути на даний контейнер, інтерфейс побудови запиту заповниться формою, для побудови обраного типу підзапиту. Якщо заповнити форму для підзапиту і

повернутися на рівень вверх, контейнер міститиме згенерований код підзапиту, як показано на рисунку 3.6. Контрольні кнопки зі стрілками дозволяють обмінювати сусідні елементи місцями, відповідно до напряму стрілок, а червона кнопка відповідає за видалення елемента – цей функціонал є аналогічним для всіх елементів системи, де зустрічаються дані контрольні кнопки.

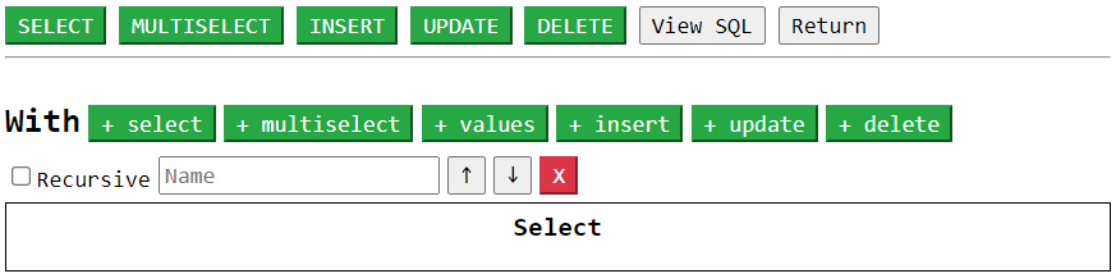


Рисунок 3.5 – пустий блок WITH-підзапиту

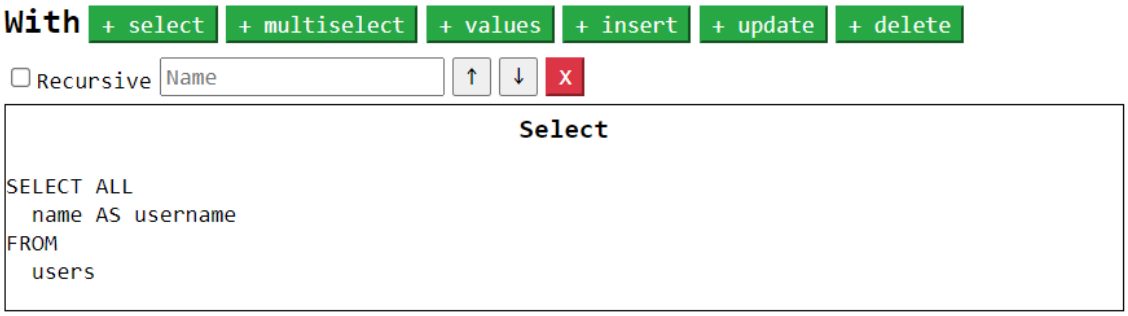


Рисунок 3.6 – Заповнений блок WITH-підзапиту

Якщо натискати на кнопки із символом «+», у відповідних частинах форми будуть додаватися нові рядки для введення запитів: SELECT – виразу задання повернутих даних, GROUP BY – вирази групування даних, ORDER BY – вирази сортування даних – всі вони показані і відповідно пронумеровані на рисунку 3.7. Кожен SELECT-елемент дозволяє додатково задати псевдонім повернутим даним, а ORDERBY-елемент дозволяє обрати напрямок сортування у випадаючому списку.

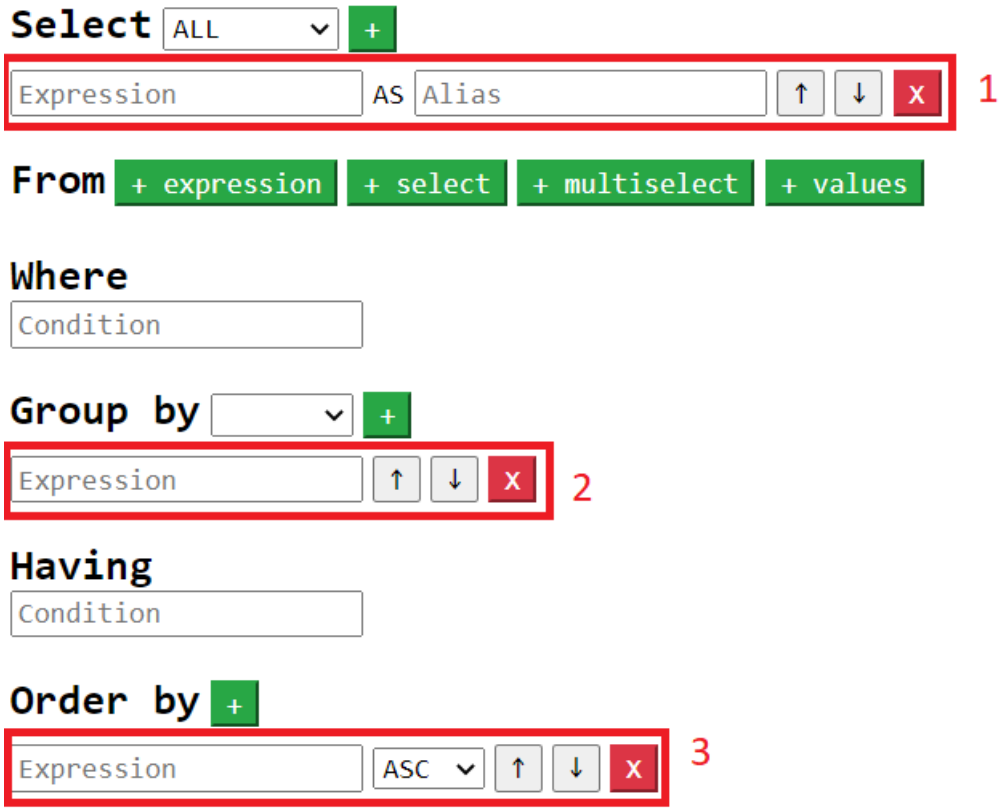


Рисунок 3.7 – Частина інтерфейсу побудови SELECT-запиту із доданими полями для вводу виразів

Якщо користувач натискатиме на кнопки у FROM-частині форми, додаватимуться блоки задання джерел даних. На рисунку 3.8 показано такі блоки: блок вводу виразу, доданий кнопкою «+ expression» і позначений цифрою 1, а також блок створення підзапиту, доданий кнопкою «+ select» і позначений цифрою 3. Оскільки було додано більше ніж одне джерело даних, між ними присутній ще один блок, позначений цифрою 2, який дозволяє обрати тип об’єднання у випадаючому списку, встановити прапорець lateral і ввести вираз-умову для об’єднання.

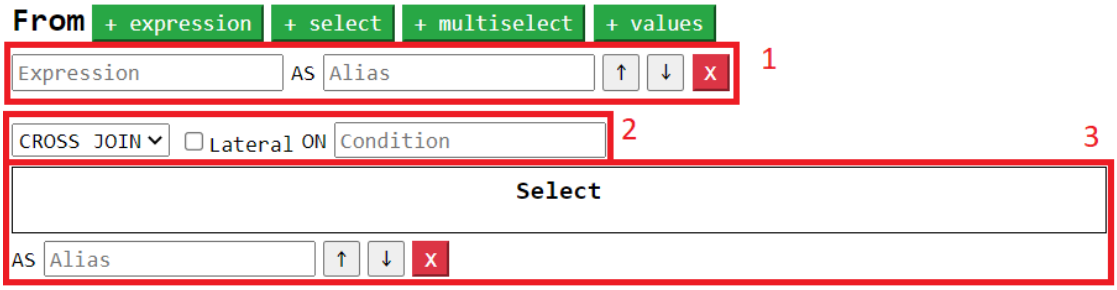


Рисунок 3.8 – FROM-блок запиту

Якщо користувач натисне на кнопку «MULTISELECT» в інтерфейсі побудови запиту, форма аналогічно заповниться елементами для побудови MULTISELECT-запиту, як показано на рисунку 3.9. Такий тип запиту не існує у мові SQL, але наявний у даному застосування, аби можна було поєднувати кілька SELECT-запитів інструментами реляційної алгебри: UNION, INTERSECT та EXCEPT. Єдиний новий елемент тут, це кнопки додавання підзапитів, позначені цифрою 1.

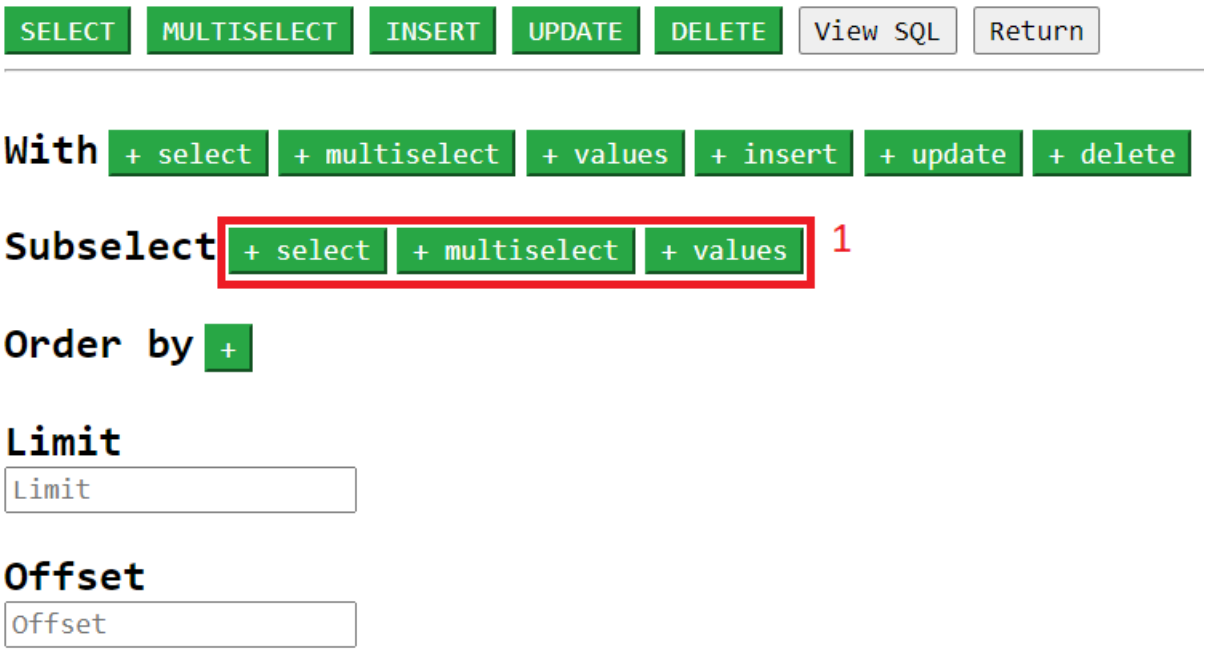


Рисунок 3.9 – Інтерфейс побудови MULTISELECT-запиту

Якщо користувач натискатиме на ці кнопки, нижче будуть додаватися блоки SUBSELECT-елементів, показані на рисунку 3.10. Якщо таких блоків більше ніж один, між ними будуть додаватися випадаючі списки, для вибору операції реляційної алгебри, котра має бути застосована.

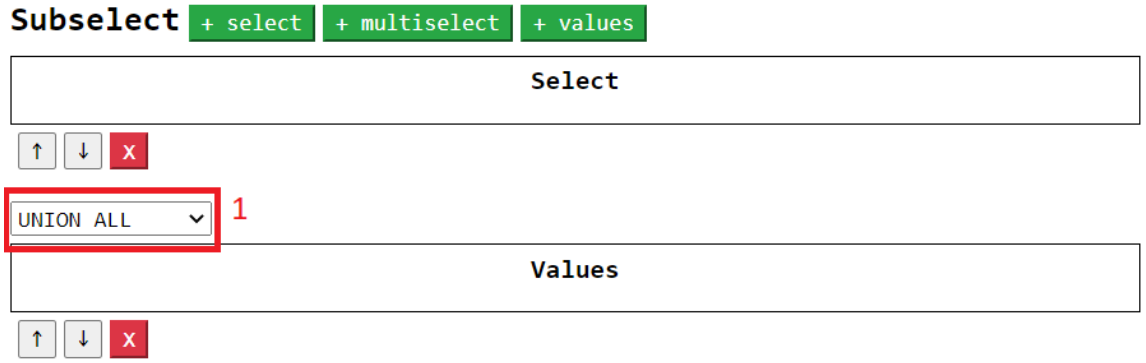


Рисунок 3.10 – SUBSELECT-блоки

На попередньому рисунку показаний VALUES-блок. У мові SQL VALUES-елемент може існувати лише як джерело даних для іншого запиту. Інтерфейс його побудови показаний на рисунку 3.11. Спочатку інтерфейс містить лише дві кнопки, позначені цифрою 1. Натискаючи їх, користувач може додавати нові колонки та рядки відповідно – результат додавання двох колонок і двох рядків позначено цифрою 2. При цьому, дані можна буде вводити, якщо і кількість колонок, і кількість рядків буде ненульовою, тому кожну з кнопок треба натиснути щонайменше один раз.

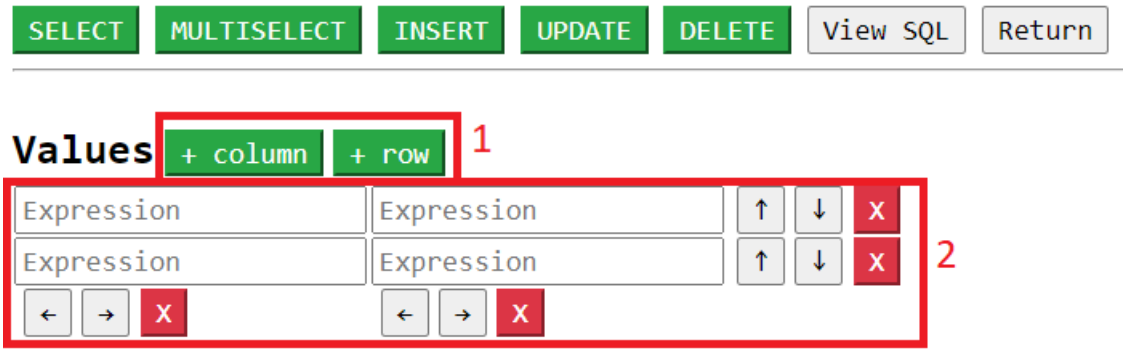


Рисунок 3.11 – Інтерфейс побудови VALUES-елемента

Якщо користувач натисне на кнопку «INSERT» в інтерфейсі побудови запиту, форма аналогічно заповниться елементами для побудови INSERT-підзапиту. Новими серед них є:

1) блок задання цілі вставки, який містить: поля назви ти псевдоніма таблиці і кнопку для додавання цільових колонок;

2) випадаючий список джерела даних;

3) випадаючий список режиму вирішення конфлікту;

4) кнопку додавання повернутих значень.

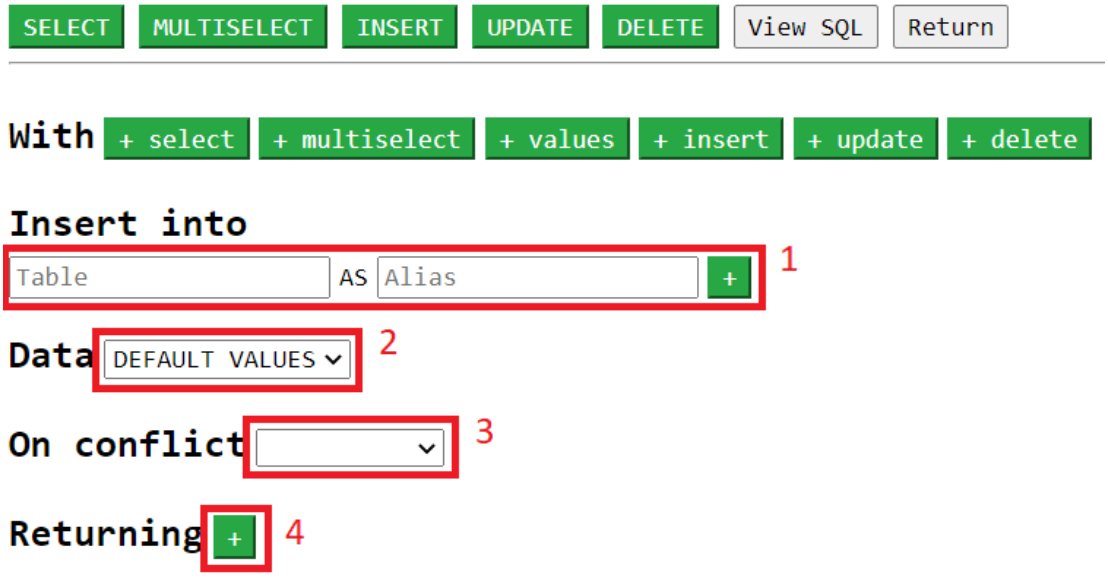


Рисунок 3.12 – Базовий інтерфейс побудови INSERT-запиту

На рисунку 3.13 показано частину цього інтерфейсу, після взаємодії із всіма її елементами. Якщо користувач натискатиме на кнопки «+», додаватимуться вирази для задання цільових колонок таблиці, помічені цифрою 1, і вирази задання повернутих даних разом із їх псевдонімами, помічені цифрою 4. Якщо у випадаючому списку джерела даних обрати будь-який елемент окрім «DEFAULT VALUES», буде додано блок задання джерела даних, помічений цифрою 2. Якщо у випадаючому списку режиму вирішення конфлікту обрати варіант «UPDATE», з'являться дві групи елементів: чотири кнопки для обрання джерел даних і два текстові поля – одне для задання цільової таблиці, а інше для задання умови конфлікту. Натискаючи на вищезгадані кнопки, користувач може додавати джерела даних, котрі будуть використані для оновлення даних. Окрім самого джерела користувач має задати перелік колонок, які будуть оновлені – для цього використовується кнопка «+ field» розташована поруч. На рисунку 3.14 показано два джерела даних разом із пов'язаними з ними переліками колонок – до кожного з переліків було додано по одній колонці.

Рисунок 3.13 – Інтерфейс побудови INSERT-запиту із доданими елементами

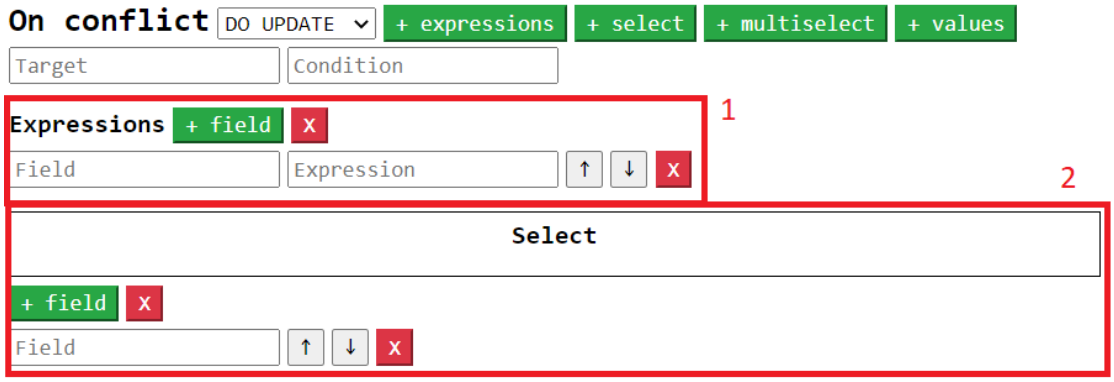


Рисунок 3.14 – ONCONFLICT-групи елементів

Якщо користувач натисне на кнопку «UPDATE» в інтерфейсі побудови запиту, форма аналогічно заповниться елементами побудови UPDATE-запиту, як показано на рисунку 3.15. Єдиним новим елементом є частина UPDATE, де користувач задає цільову таблицю та її псевдонім, помічена цифрою 1. Частина SET, помічена цифрою 2, відповідає елементам задання джерел даних у частини ON CONFLICT INSERT-запиту, коли користувач обирає режим вирішення конфлікту «DO UPDATE».

Якщо користувач натисне на кнопку «DELETE» в інтерфейсі побудови запиту, форма аналогічно заповниться елементами побудови DELETE-запиту, як показано на рисунку 3.16. Єдиним новим елементо є частина DELETE, де користувач задає цільову таблицю та її псевдонім, помічена

цифрою 1. Частина USING є аналогічною частині FROM для SELECT- і UPDATE-запитів.

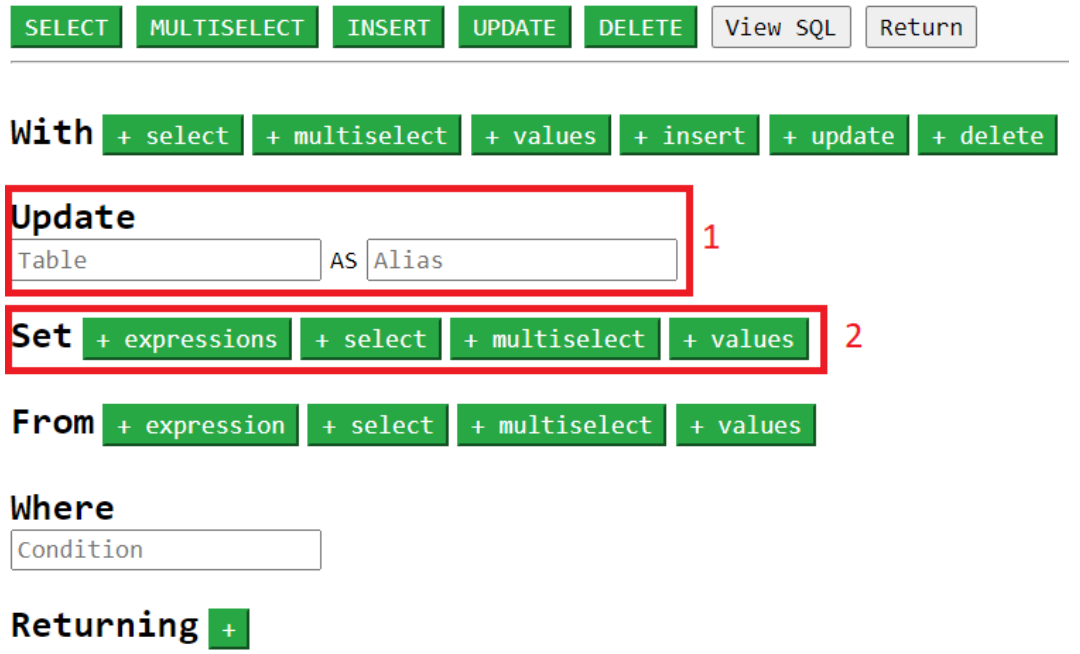


Рисунок 3.15 – Інтерфейс побудови UPDATE-запиту

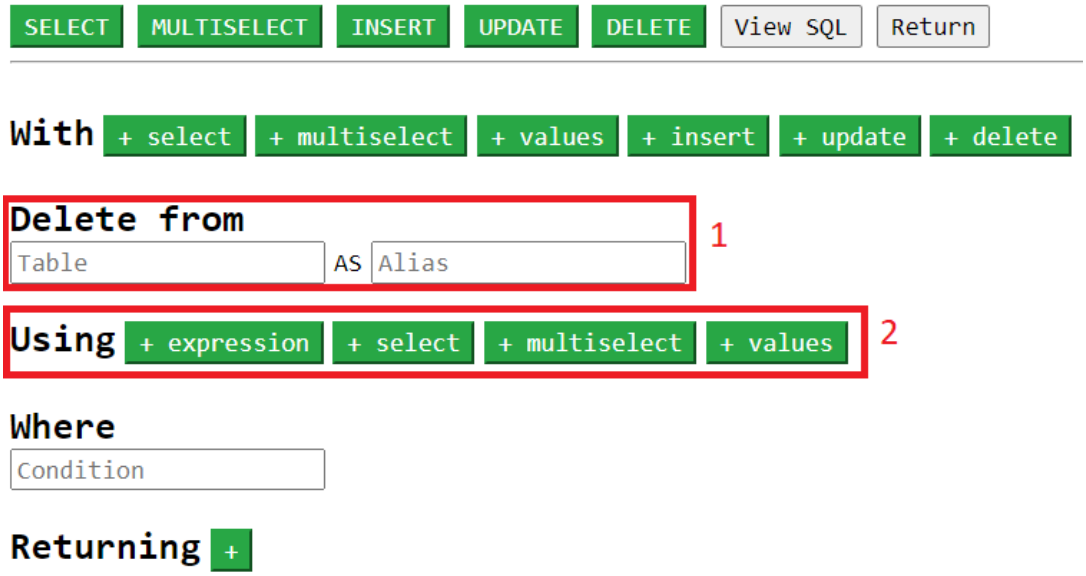


Рисунок 3.16 – Інтерфейс побудови DELETE-запиту

На рисунку 3.17 зображено довільний правильно побудований SELECT-запит, не рахуючи полів, розташованих у формі побудови запиту знизу, котрі не редагувалися. Якщо користувач натисне на кнопку «VIEW SQL», відкриється вікно перегляду згенерованого SQL коду, показане на рисунку 3.18.

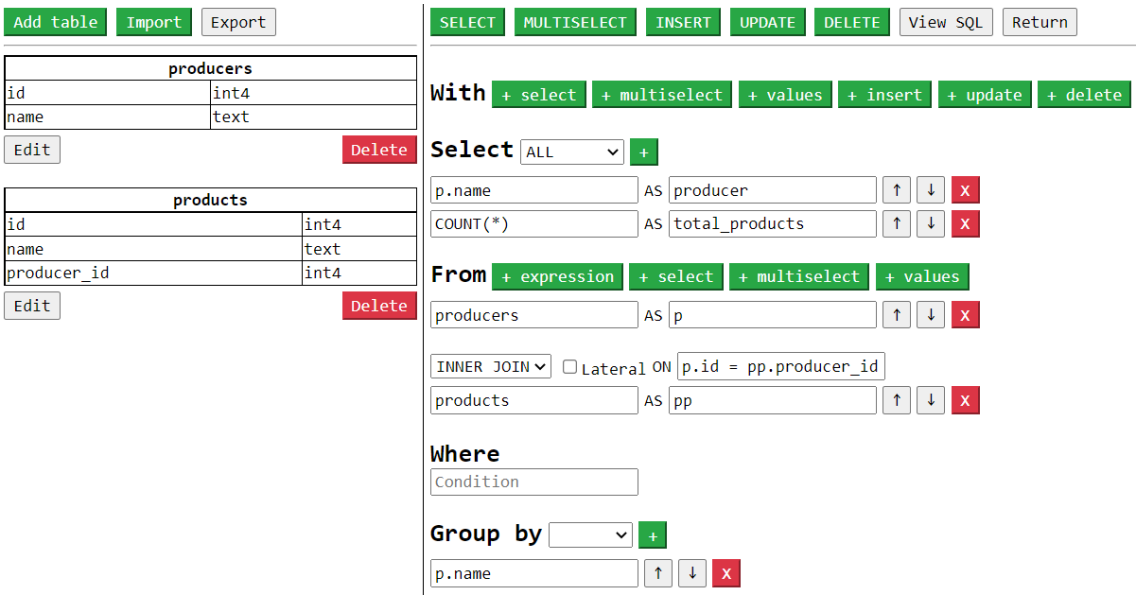


Рисунок 3.17 – Приклад правильно побудованого SELECT-запиту

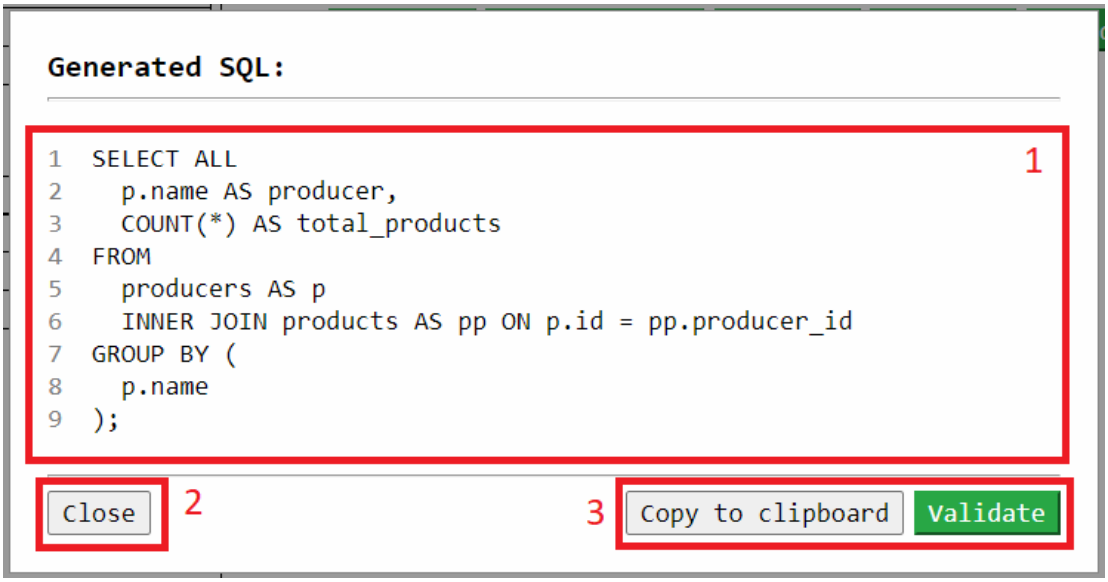


Рисунок 3.18 – Вікно перегляду згенерованого SQL коду

Дане вікно містить:

1) поле, зі згенерованим SQL кодом;
2) кнопку для закриття цього вікна;
3) кнопки копіювання коду та його валідації.

Якщо користувач натисне на кнопку копіювання, код буде скопійовано до буфера обміну. Якщо користувач натисне на кнопку валідації, за наявності підключення до сервера, відбудеться валідація згенерованого коду, а результат буде виведено всередині вікна. На рисунку 3.19 зображено приклад успішної валідації коду, повідомлення про успіх позначено цифрою 1. На

рисунку 3.20 зображено приклад невдалої валідації коду, після внесення змін у інтерфейсі побудови запиту, із метою викликати помилку.

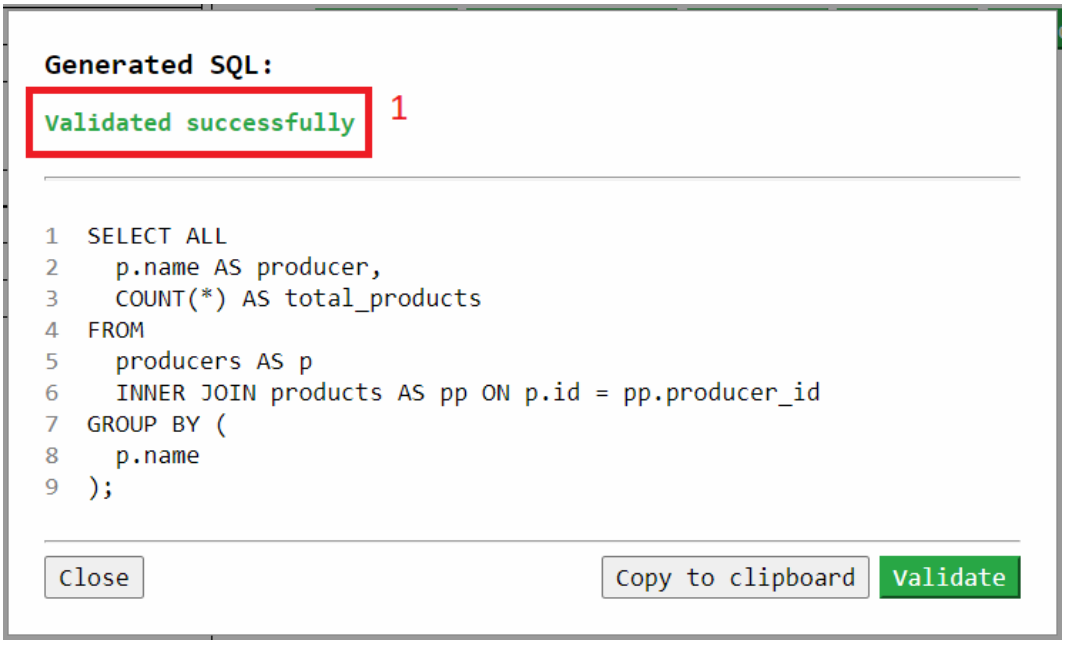


Рисунок 3.19 – Приклад успішної валідації коду

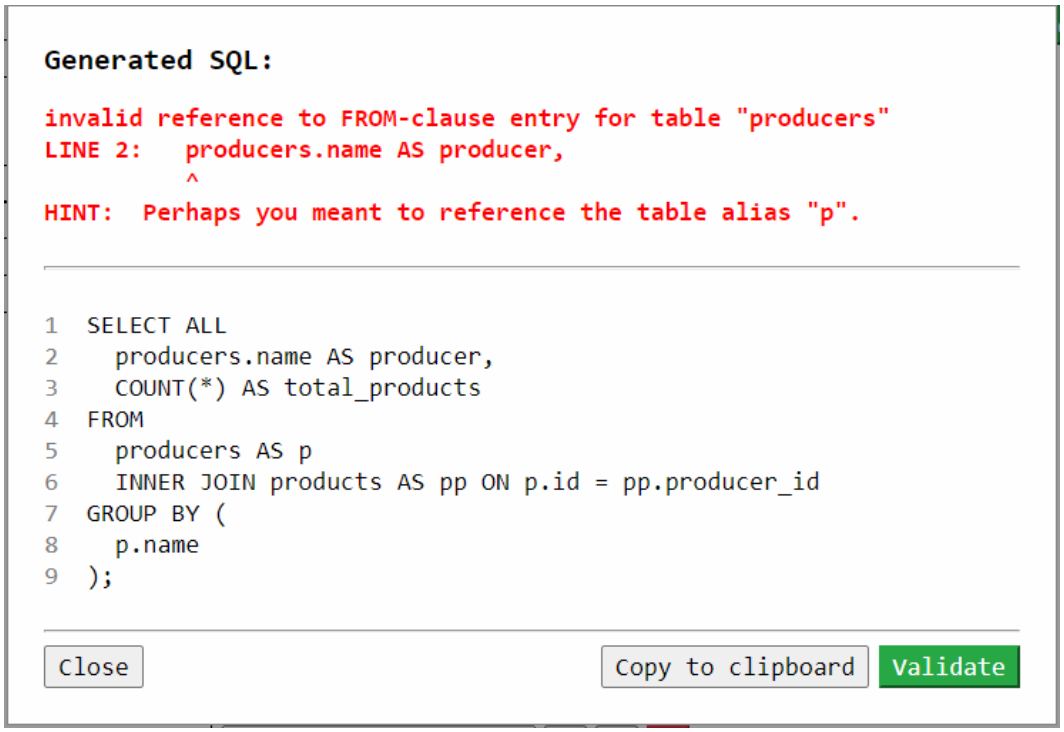


Рисунок 3.20 – Приклад невдалої валідації коду

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2022 р.

Веб-застосування для побудови запитів до бази даних PostgreSQL із використанням графічного інтерфейсу

Графічний матеріал

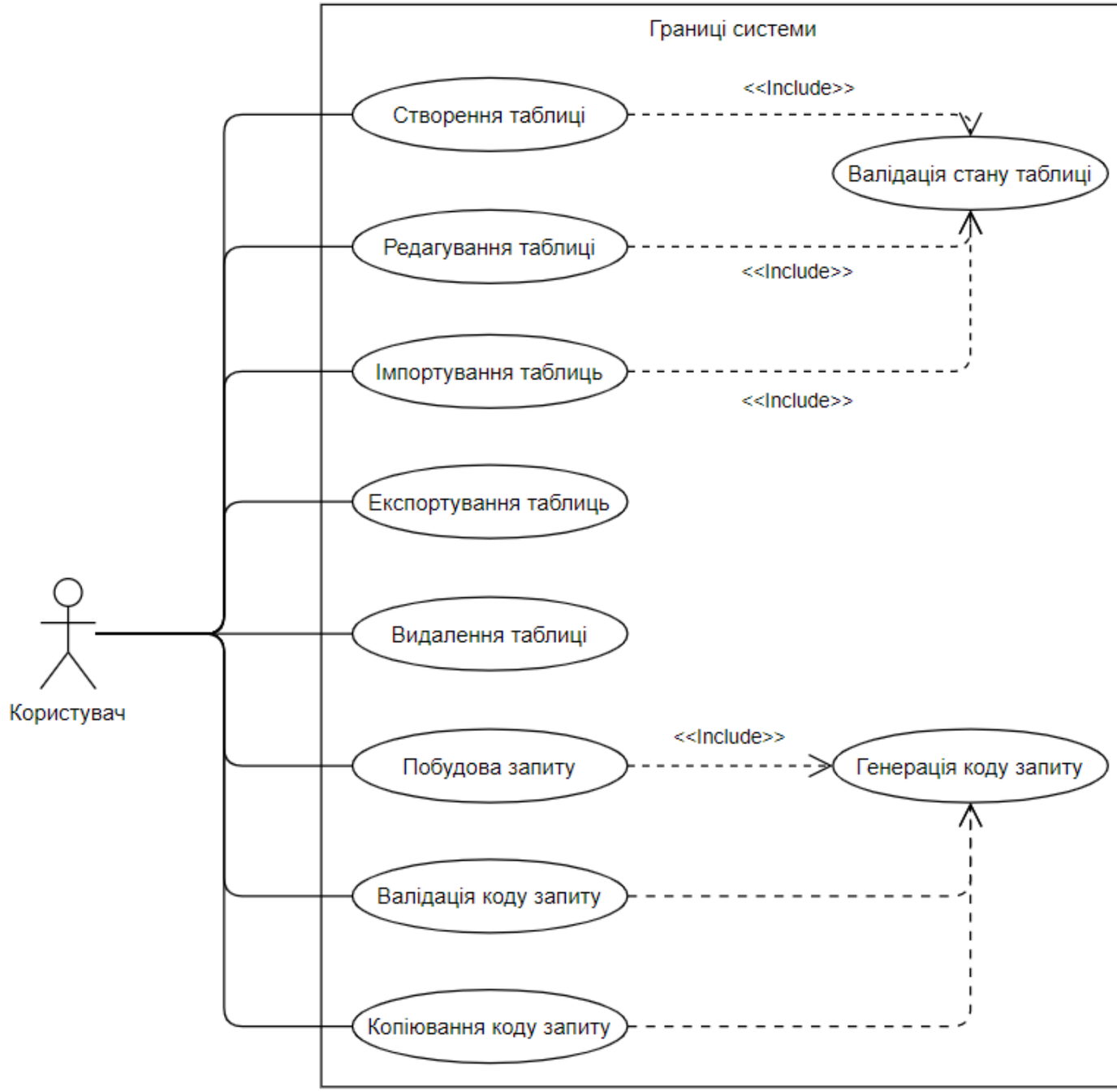
КПІ.ІП-8135.045300.06.99

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Катерина ЛІЩУК

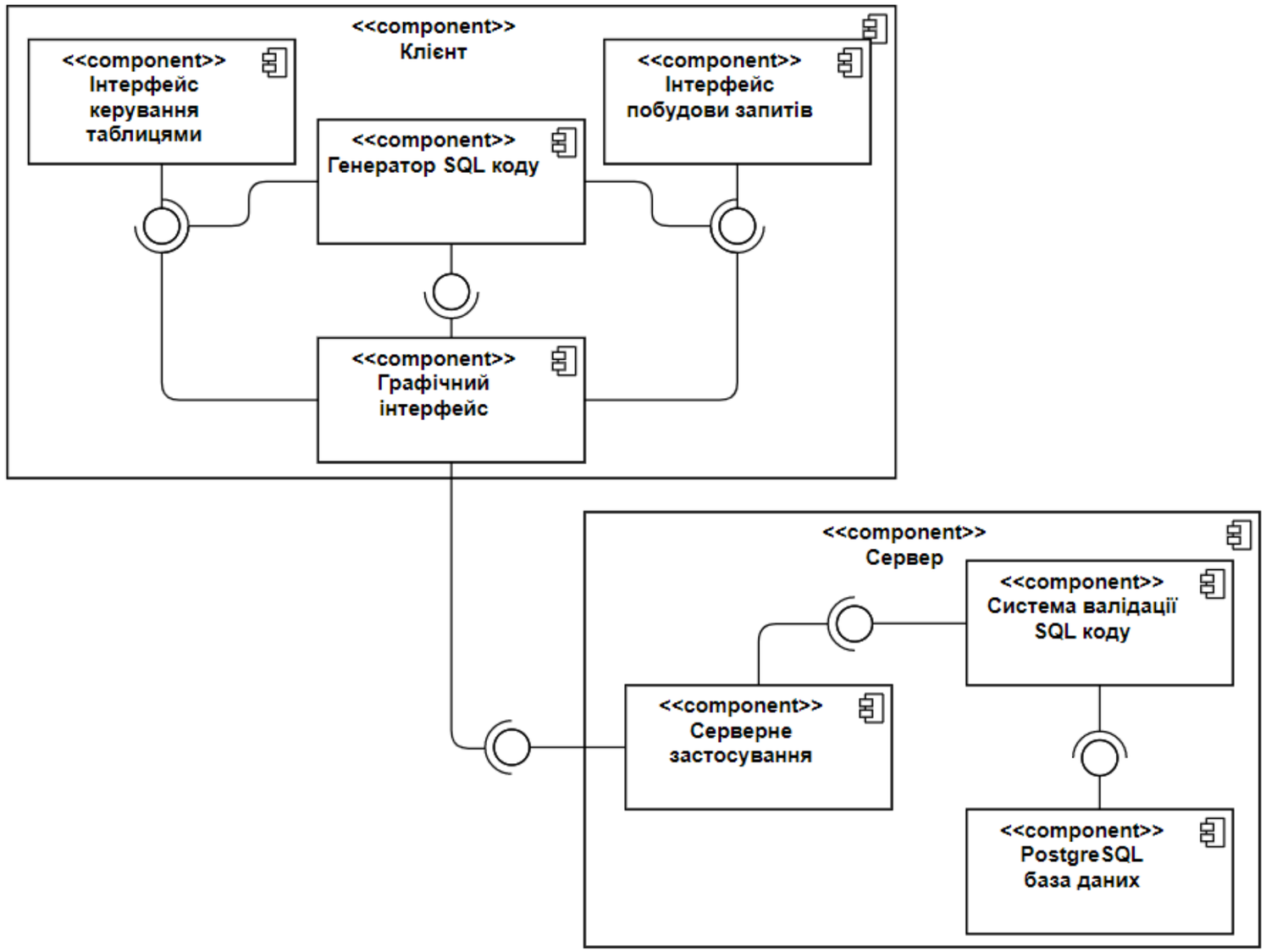
Нормоконтроль:
_____ Катерина ЛІЩУК

Виконавець:
_____ Олександр ФЕДОРОВ

Київ – 2022



					КПІ.ІП-8135.045300.06.99.СС		
					Схема структурна варіантів використання	Літера	Маса / Масштаб
Зм.	Арк.	№ документа	Підпис	Дата			
Розробив		Федоров О.В.				Аркуш	Аркушів
Перевірив		Ліщук К.І.					
Т. кон.							
					Веб-застосування для побудови запитів до бази даних PostgreSQL із використанням графічного інтерфейсу	КПІ ім. Ігоря Сікорського ФІОТ каф. ІПІ гр. ІП-81	
Н. кон.		Ліщук К.І.					
Затвердив		Ліщук К.І.					



					КПІ.ІП-8135.045300.06.99.СС		
					Схема структурна компонентів програмного забезпечення	Літера	Маса / Масштаб
Зм.	Арк.	№ документа	Підпис	Дата			
Розробив		Федоров О.В.				Аркуш	Аркушів
Перевірив		Ліщук К.І.					
Т. кон.					Веб-застосування для побудови запитів до бази даних PostgreSQL із використанням графічного інтерфейсу	КПІ ім. Ігоря Сікорського ФІОТ каф. ІПІ гр. ІП-81	
Н. кон.		Ліщук К.І.					
Затвердив		Ліщук К.І.					

Зм.	Арк.	№ документа	Підпис	Дата		
Розробив		Федоров О.В.			Кресленяя виглядУ екранних форм	Літера	Маса	Масштаб
Перевірив		Лішук К.І.						
Т. кон.						Аркуш	Аркушів	
Н. кон.		Лішук К.І.			Веб-застосування для побудови запитів до бази даних PostgreSQL із використанням графічного інтерфейсу	КПІ ім. Ігоря Сікорського ФІОТ каф. ІПІ гр. ІП-81		
Затвердив		Лішук К.І.						

КПІ.ІП-8135.045300.06.99.КЕ

КПІ.ІП-8135.045300.06.99.КЕ

Add table Import Export

users

id	int4
name	text
date_of_birth	time
email	text

Edit Delete

products

id	int4
name	text
price	numeric

Edit Delete

orders

id	int4
user_id	int4
product_id	int4
quantity	int4

Edit Delete

SELECT MULTISELECT INSERT UPDATE DELETE View SQL Return

With + select + multiselect + values + insert + update + delete

Select ALL + +

p.name AS product ↑ ↓ X

SUM(o.quantity) AS quantity ↑ ↓ X

From + expression + select + multiselect + values

products AS p ↑ ↓ X

INNER JOIN lateral ON p.id = o.product_id

orders AS o ↑ ↓ X

Where

Condition

Group by + +

p.name ↑ ↓ X

Having

Condition

Order by +

Limit

Limit

Offset

Offset