Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)
Кафедра Штучного інтелекту
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Візуальний генератор пошукових запитів NL-to-SQL для проєктування РБД
(тема)

Виконав:
здобувач четвертого року навчання,
групи ІТШІ-21-3
Ірина Кашпур
(власне ім'я, прізвище)
Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Штучний інтелект
(повна назва освітньої програми)
Керівник ст. викл. Олена Гриньова
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ШІ _____ Олег ЗОЛОТУХІН
(підпис) (власне ім'я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Штучного інтелекту

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Штучний інтелект
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____« _____ 20 ___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Кашпур Ірині Вікторівні
(прізвище, ім'я, по батькові)

1. Тема роботи Візуальний генератор пошукових запитів NL-to-SQL для проєктування РБД

затверджена наказом університету від 19 травня 20 25 р. № 378Ст
2. Термін подання студентом роботи до екзаменаційної комісії 17 червня 20 25 р.
3. Вихідні дані до роботи Науково-технічні публікації, дані Інтернет-джерел, інструменти для створення ER-діаграм, практичні бібліотеки Java-Script, навчальні набори данних.

4. Перелік питань, що потрібно опрацювати в роботі
1) Аналіз предметної галузі та постановка задачі
2) Обґрунтування використаних технології та методів
3) Архітектура системи та програмна реалізація

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	19.05.2025	виконано
2	Аналіз предметної галузі	19.05.2025 – 20.05.2025	виконано
3	Актуальність проблеми візуальної генерації	21.05.2025	виконано
	схеми БД		
4	Мета і постановка задачі	22.05.2025	виконано
5	Огляд та порівняльний аналіз існуючих рішень	22.05.2025 – 23.05.2025	виконано
5	Аналіз методів, бібліотек та наборів даних	23.05.2025 – 24.05.2025	виконано
6	Опис програмної реалізації	24.05.2025 – 25.05.2025	виконано
7	Оформлення пояснювальної записки	25.05.2025 – 30.05.2025	виконано
8	Підготовка до захисту	01.06.2025 – 16.06.2025	виконано
9	Захист перед ЕК	17.06.2025	виконано

Дата видачі завдання 19 травня 20 25 р.

Здобувач _____
(підпис)

Керівник роботи _____ ст. викл. Олена Гриньова
(підпис) (посада, власне ім’я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 57 с., 26 рис., 1 табл., 1 дод., 15 джерел.

АРХІТЕКТУРА СИСТЕМИ, ВІЗУАЛЬНИЙ ГЕНЕРАТОР, ГЕНЕРАЦІЯ SQL-КОДУ, ГРАФІЧНИЙ ІНТЕРФЕЙС, ШТУЧНИЙ ІНТЕЛЕКТ, CSS, ER-ДІАГРАМА, HTML, JAVASCRIPT, PYTHON, SQL.

Мета роботи – розробка візуального генератора пошукових запитів NL-to-SQL для проектування реляційних баз даних (РБД), який інтегрує дві моделі штучного інтелекту та дозволяє користувачам легко створювати або генерувати таблиці за допомогою ШІ, автоматично реалізуючи SQL-код.

Застосунок має оптимізувати процес проєктування БД, забезпечуючи скорочення часу на створення ER-діаграм та SQL-коду, підвищення точності, оптимізацію SQL-запитів та інтеграцію технологій штучного інтелекту для створення SQL-коду на основі текстових описів.

Об’єктом дослідження у даній роботі є система автоматизованого проєктування та генерації ER-діаграм для РБД, що реалізована як веб-інтерфейс із використанням сучасних технологій програмування.

Предметом дослідження виступають методи, алгоритми та програмні засоби, що забезпечують інтуїтивну побудову ER-діаграм, автоматичну генерацію SQL-коду, а також організацію безпечної взаємодії користувача з системою. Особливу увагу приділено розробці функціонального інтерфейсу, логіці побудови зв’язків між сутностями, реалізації перевірки даних, підтримці різних типів ключів, унікальних груп та забезпеченню збереження і відновлення даних користувача.

Методологія дослідження базувалася на інтеграції сучасних технологій для створення автоматизованого генератора ER-діаграм і SQL-коду. Експериментальний підхід включав тестування інструменту серед користувачів для оцінки його ефективності, зручності та точності.

ABSTRACT

Bachelor’s thesis contains: 57 pp., 26 fig., 1 tabl., 1 ann., 15 references.

ARTIFICIAL INTELLIGENCE, CSS, ER-DIAGRAM, GRAPHICAL USER INTERFACE, HTML, JAVASCRIPT, PYTHON, SQL, SQL CODE GENERATION, SYSTEM ARCHITECTURE, VISUAL GENERATOR.

The aim of this work is to develop a visual query generator, NL-to-SQL, for designing relational databases (RDB), which integrates two artificial intelligence models and enables users to easily create or generate tables using AI, automatically implementing the corresponding SQL code. The application is intended to optimize the database design process by reducing the time required to create ER diagrams and SQL code, improving accuracy, optimizing SQL queries, and integrating AI technologies for generating SQL code based on textual descriptions.

The object of the study in this work is an automated system for designing and generating ER diagrams for RDBs, implemented as a web interface using modern programming technologies. The subject of the research includes methods, algorithms, and software tools that provide intuitive ER diagram construction, automatic SQL code generation, as well as organization of secure user interaction with the system. Special attention is given to the development of a functional interface, the logic of establishing relationships between entities, implementation of data validation, support for various types of keys, unique groups, and ensuring data preservation and recovery for users.

The research methodology is based on the integration of modern technologies to create an automated generator of ER diagrams and SQL code. The experimental approach included testing the tool among users to evaluate its effectiveness, usability, and accuracy.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД – база даних;

ВГБД – візуальний генератор схем для реляційних баз даних;

ЕР-діаграма – Entity-Relationship діаграма;

ПМ – природна мова;

РБД – реляційна база даних;

СКБД – система керування базами даних;

ШІ – штучний інтелект;

AI – Artificial Intelligence – штучний інтелект;

API – Application Programming Interface – прикладний програмний інтерфейс;

DDL – Data Definition Language – мова визначення даних;

DML – Data Manipulation Language – мова маніпулювання даними;

DUI – Dedicated User Interface – спеціалізований користувацький інтерфейс;

ER – Entity-Relationship – сутність-зв'язок;

ML – Machine Learning – машинне навчання;

NL2SQL – Natural Language to SQL – перетворення природної мови в SQL;

SQL – Structured Query Language – мова структурованих запитів.

ВСТУП

ER-діаграма є інструментом, що забезпечує візуалізацію сукупності знань про структуру предметної галузі і слугує початковим етапом для генерації SQL-коду, що реалізує цю структуру у конкретній СУБД.

Проєктування баз даних є важливим етапом у розробці інформаційних систем. В даний час для проєктування всього проекту «з нуля» необхідний експерт, який визначить предметну галузь, створить структуру (схему), ключові таблиці, їх атрибути, тощо та реалізує все в коді SQL.

За допомогою класичних методів аналізу та документування, ER-моделі можуть бути створені повністю вручну, що вимагає значних зусиль та часу, а також самостійного написання SQL-коду, що підвищує ризик помилок та створення неефективних запитів. Розробники постійно стикаються з необхідністю створення схожих структур, таких як схеми аутентифікації чи каталоги продуктів, замість повторного використання готових рішень.

У світі сучасних технологій виникає питання, чи можливо забезпечити безперервний, миттєвий перехід від концептуального моделювання до фізичного створення бази даних, а також чи можна розробити ефективне, інтуїтивно зрозуміле та зручне середовище для побудови ER-діаграм, яке б відповідало сучасним вимогам розробки. Важливо також оцінити, наскільки розроблене середовище буде актуальним і корисним у контексті реального процесу створення та супроводу баз даних, враховуючи динамічність технологічних змін.

У цій роботі детально описано розробку графічного інтерфейсу для побудови ER-діаграм із застосуванням сучасних технологій Web, Python та SQL та ключові спостереження, отримані у процесі. Особлива увага приділяється створенню автоматизованої системи перекладу ER-моделей на різні мови запитів, що розширює функціональні можливості інструменту та підвищує його універсальність при проектуванні БД.

У роботі аналізуються як технічні, так і функціональні аспекти розробленої системи, що дозволяє оцінити її значення при моделюванні даних.

Для розробки візуального генератора необхідно мати ґрунтовні знання щодо процесів проектування та побудови БД. Це також передбачає розуміння теоретичних основ створення ER-діаграм, їх ключових цілей та переваг, які вони можуть забезпечити при правильному застосуванні. Знання про побудову ER-діаграм було отримано безпосередньо в процесі реалізації проекту, а також шляхом вивчення необхідних елементів, таких як логіка встановлення зв’язків між сутностями та їх подальша підтримка. Було сформувано цілісне уявлення про механізми моделювання, що є основою для ефективного створення БД.

Додатково, у звіті розглядається оцінка якості макета БД, створеного за допомогою розробленої системи, з використанням інтелектуальних систем рекомендацій, які базуються на ШІ з контекстним розумінням. Такий підхід дозволяє не лише автоматизувати процес проєктування, але й підвищити адаптивність створеного макета відповідно до специфіки предметної галузі та вимог користувача. В результаті, було встановлено, що створена програмна система займає вагоме місце в галузі проектування БД, демонструючи практичну цінність та наукову новизну.

Дослідження спрямоване на комплексний аналіз ролі та значення графічного інтерфейсу конструктора ER-діаграм у процесі проектування БД, а також на розробку інструментарію, який поєднує автоматизацію, багатомовність, інтелектуальні рекомендації та зручність використання.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Основні підходи до побудови ER-діаграм

Модель «сутність-зв'язок» (ER-модель) описує структуру БД за допомогою діаграми. ER-модель – це концептуальна модель предметної галузі, яка може бути реалізована в системі керування базами даних. Побудова ER-діаграм є важливим етапом аналізу предметної галузі та процесу концептуального моделювання баз даних. Саме ER-діаграми дозволяють формалізувати структуру даних, встановити взаємозв’язки між сутностями, визначити атрибути та ключові характеристики даних та відношень, що забезпечує коректне проєктування логічної та фізичної моделі БД.

Традиційний підхід до побудови ER-діаграм був запропонований Пітером Ченом у 1976 році [1]. Згідно з його моделлю, основними елементами ER-діаграми є сутності (entities), зв’язки (relationships) та атрибути (attributes). Сутність визначає об’єкт предметної галузі, який має унікальні характеристики. Зв’язки формалізують взаємодію між сутностями. Атрибути описують властивості як сутностей, так і зв’язків. Важливою частиною є визначення типу зв’язку – «один-до-одного», «один-до-багатьох» або «багато-до-багатьох».

На практиці побудова ER-діаграм починається з визначення основних сутностей предметної галузі, після чого визначаються їх атрибути та ключі (первинний, зовнішній, унікальні групи). Далі встановлюються зв’язки між сутностями, які можуть мати власні атрибути, якщо вони є асоціативними. Нарешті, перевіряють, щоб усе відповідало правилам цілісності даних, наприклад, щоб не було дублікатів ключів. Для створення діаграм можна малювати їх від руки, але частіше використовують автоматизовані засоби, як-от ERwin чи Draw.io. Такі інструменти

дозволяють зменшити кількість помилок та підвищити швидкість проєктування.

У сучасних дослідженнях розрізняють два основних підходи для побудови ER-діаграм: геометричний та семантичний. Геометричний підхід орієнтований на візуальне компонування діаграми, де основна увага приділяється зручності сприйняття та компактності розташування елементів. Семантичний підхід зосереджений на коректному відображенні сутностей, зв’язків та обмежень цілісності, що забезпечує точність моделі та її відповідність реальній структурі даних.

1.2 Сучасний стан автоматизованого проєктування БД

Автоматизоване проєктування БД є однією з тенденцій сучасної інформатики, що визначає ефективність, масштабованість та надійність інформаційних систем у різних сферах діяльності. Зі зростанням обсягів і різноманіття даних, а також із поширенням концепцій цифрової трансформації, питання автоматизації процесів проєктування БД набуває особливої актуальності.

Сучасні інструменти автоматизованого проєктування надають потужний функціонал для побудови ER-діаграм у графічному режимі. Останні роки характеризуються появою розширених підходів до побудови ER-діаграм, які враховують специфіку гібридних систем (реляційних і NoSQL), підтримку складних типів зв’язків, багаторівневих ієрархій та атрибутів з вкладеними структурами. Важливим напрямом є також автоматична нормалізація моделі, аналіз надлишкових залежностей та оптимізація структури для підвищення продуктивності БД. Сучасні програмні засоби забезпечують інтуїтивно зрозумілий графічний інтерфейс, автоматичну генерацію SQL-коду для різних СКБД, а також перевірку коректності моделі на етапі проектування. Важливою особливістю цих систем є підтримка drag-and-drop інтерфейсів, автоматична генерація

атрибутів та зв’язків, валідація моделі, що дозволяє швидко створювати та редагувати структуру БД, а також інтеграція з хмарними сервісами для збереження роботи та підтримки версій проєктів.

Окрему увагу сучасні дослідження приділяють питанням оптимізації структури БД на основі аналізу типових сценаріїв використання. Алгоритми машинного навчання пропонують автоматично індексацію та зміну типів даних для підвищення продуктивності системи в умовах великих обсягів інформації. Такі підходи вже реалізовані у провідних хмарних платформах (Amazon RDS, Google Cloud SQL), що дозволяє автоматизувати не лише проєктування, а й обслуговування БД.

Автоматизація проєктування БД стикається також з низкою викликів. Серед них – необхідність забезпечення безпеки даних, відповідності стандартам захисту персональної інформації (GDPR), підтримки гібридних моделей даних, а також інтеграції з існуючими корпоративними системами. Важливим аспектом є також забезпечення гнучкості та адаптивності інструментів автоматизації, що дозволяє враховувати специфіку різних предметних галузей.

Таким чином, сучасний стан автоматизованого проєктування БД характеризується переходом від ручних, трудомістких процесів до інтелектуальних, інтегрованих рішень, що забезпечують високу продуктивність, надійність та адаптивність інформаційних систем. Розвиток цієї галузі пов’язаний із впровадженням глибокого навчання для передбачення вимог до структури БД, розробкою кросплатформних рішень для автоматизації міграції між різними СКБД та забезпеченням повної інтеграції з хмарними сервісами.

1.3 Аналіз інструментів для генерації SQL-коду

Автоматизована генерація SQL-коду стала важливою складовою сучасних систем проєктування БД. На ринку існує широкий вибір

програмних рішень, які реалізують даний функціонал, і кожне з них має свої особливості, переваги та недоліки. Проведемо аналіз найбільш поширених інструментів, які використовуються як у промислових, так і в наукових проєктах (таблиця 1.1).

Таблиця 1.1 – Порівняльний аналіз

Інструмент	Підтримка СКБД	Візуалізація	Спільна робота	Генерація SQL	AI-функції	Вартість
MySQL Workbench	MySQL	+	-	+	-	Безкоштовно
dbdiagram.io	MySQL, PostgreSQL, SQL Server, SQLite	+	+	+	-	Платно/ Безкоштовно
Lucidchart	Різні СКБД	+	+	+	-	Платно
DrawSQL	Популярні СКБД	+	+	+	-	Платно
ChatGPT, Copilot	Різні СКБД	-	-	+	+	Платно/ Безкоштовно

MySQL Workbench є одним із найпопулярніших інструментів для візуального проєктування БД, який має вбудований механізм генерації SQL- коду для створення, зміни та видалення об’єктів БД. Перевагою Workbench є його інтеграція з MySQL Server, підтримка реверсивного інжинірингу (Reverse Engineering) та прямого експорту SQL-скриптів. Інтерфейс дозволяє користувачу створювати ER-діаграми, які автоматично трансформуються у DDL-скрипти. Проте, інструмент орієнтований переважно на MySQL і має обмежену підтримку інших СКБД.

dbdiagram.io – це веб-інструмент для побудови ER-діаграм із можливістю генерації SQL-коду для різних СКБД (MySQL, PostgreSQL, SQL Server, SQLite). Його особливістю є простота використання, діаграми

можна створювати як у графічному режимі, так і за допомогою текстового DSL (Domain-specific Language). Інструмент дозволяє швидко експортувати структуру у вигляді SQL-скриптів, що робить його зручним для командної роботи та швидкого прототипування. Недоліком є обмежена підтримка складних обмежень цілісності та відсутність глибокої інтеграції з локальними СКБД.

Lucidchart і DrawSQL – це хмарні сервіси для візуального моделювання БД із генерацією SQL-коду. Вони підтримують спільну роботу над проєктами, керування версіями, інтеграцію з іншими корпоративними інструментами. Lucidchart має розширені можливості щодо кастомізації діаграм та експорту у різні формати, включаючи SQL. DrawSQL орієнтований на командну роботу та швидке створення схем із подальшим експортом у SQL-скрипти для популярних СКБД. Обидва сервіси мають обмеження у безкоштовних версіях, а також не завжди коректно підтримують специфічні типи даних та складні зв’язки.

1.4 Використання штучного інтелекту у проєктуванні схеми БД

В останнє десятиріччя штучний інтелект (ШІ) став рушійною силою трансформації інформаційних систем, зокрема – у проєктуванні та оптимізації структур баз даних [2].

Зі зростанням обсягів та різноманіття даних, традиційні підходи моделювання БД виявилися недостатньо гнучкими, особливо для складних, динамічних систем, таких як системи комп’ютерного зору, розпізнавання емоцій, біометричні або освітні аналітичні платформи. У цьому контексті ШІ виступає не лише як інструмент автоматизації, а й як засіб для підвищення адаптивності, масштабованості та інтелектуальної підтримки прийняття рішень у процесі проєктування БД [3].

Останнім часом набувають популярності інструменти, що використовують генеративні моделі штучного інтелекту для створення SQL-коду на основі текстових описів або ER-діаграм.

Наприклад, ChatGPT від OpenAI може згенерувати SQL-скрипти для різних СКБД, оптимізувати запити, автоматично виправляти помилки у коді. Подібний функціонал реалізовано і в GitHub Copilot. Перевагою таких інструментів є можливість працювати з природною мовою, що знижує поріг входу у галузь розробки БД для новачків і підвищує продуктивність досвідчених розробників. Однак, генерація коду не завжди є абсолютно коректною, особливо для складних структур, і потребує додаткової перевірки.

Один із перспективних напрямів розробки – автоматичний аналіз вимог і генерація концептуальних та логічних схем БД. Сучасні дослідження [4] демонструють, що глибокі нейронні мережі (наприклад, трансформери) здатні аналізувати текстові описи предметної галузі, виявляти сутності, атрибути, зв’язки та автоматично будувати ER-діаграми або навіть DDL-скрипти для реляційних СУБД. Це значно скорочує час, зменшує людський фактор та підвищує якість проєктування, особливо у складних галузях, де вимоги часто змінюються.

Машинне навчання застосовується для динамічної оптимізації структури БД, зокрема – для автоматичного створення, видалення чи перебудови індексів, секцій (логічних розділів), а також для вибору оптимальних типів зберігання даних [5]. Наприклад, Reinforcement Learning (RL) дозволяє системі в реальному часі аналізувати патерни запитів і адаптувати структуру БД для мінімізації часу відповіді. У хмарних та розподілених середовищах це особливо важливо для забезпечення SLA та ефективного використання ресурсів. Глибоке навчання, зокрема Autoencoders та GAN, використовується для автоматичної нормалізації схем, виявлення надлишкових або аномальних зв’язків, дублювання даних, а також для пропозицій щодо реструктуризації БД [6].

ШІ-моделі здатні не лише автоматизувати технічні аспекти, а й враховувати семантику предметної галузі. Застосування онтологій (OWL, RDF) у поєднанні з NLP-моделями дозволяє будувати глибоко семантичні структури БД, які краще відображають реальні зв’язки між об’єктами та підтримують гнучкий пошук і аналітику [7].

Міжнародна практика демонструє активне впровадження ШІ у проєктування БД у різних галузях – від біоінформатики до фінансових технологій. Провідні дослідники розробляють так звані «Self-Driving Databases», які здатні самостійно адаптувати свою структуру та параметри під поточне навантаження та типи даних [8]. Такі системи вже впроваджуються у великих корпораціях (Google, Microsoft, Amazon) та відкривають нові можливості для автоматизації та інтелектуальної підтримки прийняття рішень.

Окремо варто відзначити розвиток графових БД (наприклад, Neo4j, TigerGraph), де ШІ використовується для автоматичного виявлення та моделювання складних мережевих зв’язків, що особливо корисно для аналізу соціальних, біологічних або освітніх мереж.

Попри значні переваги, впровадження ШІ у проєктування БД потребує деяких допрацювань. Проблеми інтерпретованості – рішення, запропоновані глибокими моделями, не завжди прозорі для розробників. Для навчання моделей потрібні великі, репрезентативні та чисті датасети. Складні моделі вимагають значних ресурсів для навчання та передбачення результатів. Автоматизація обробки персональних даних вимагає дотримання стандартів захисту та етики.

Очікується, що у найближчі роки ШІ стане стандартом для проєктування БД у складних системах. Розвиватимуться розподілені системи з автоматичною оптимізацією структур у реальному часі; гібридні моделі (нейро-символьні, онтологічні), які поєднують гнучкість глибокого навчання з прозорістю символьних підходів; Self-tuning Databases –

системи, що самостійно адаптують свою структуру, індекси та політику зберігання під поточні задачі [9].

1.5 Мета та постановка задачі

Метою даної роботи є створення інтерактивного інструменту – візуального генератора пошукових запитів NL-to-SQL, який оптимізує процес проєктування РБД. Інструмент має забезпечити скорочення часу на створення ER-діаграм і SQL-коду, підвищення точності та оптимізацію запитів, а також інтеграцію технологій ШІ для генерації SQL-коду на основі текстових описів природною мовою. Особлива увага приділяється зручності використання, підтримці бібліотеки підсхем, повторному використанню рішень та автоматизації рутинних операцій.

Для досягнення поставленої мети необхідно вирішити такі основні задачі:

– аналіз предметної галузі та сучасних підходів;

– провести огляд сучасних методів автоматизованого проєктування БД, інструментів для побудови ER-діаграм і генерації SQL-коду, а також визначити актуальні проблеми у цій сфері;

– програмна реалізація та тестування системи.

2 ОБҐРУНТУВАННЯ ВИКОРИСТАНИХ ТЕХНОЛОГІЇ ТА МЕТОДІВ

У цьому розділі розглянуті як наукові, так і технічні аспекти, що лежать в основі розробки системи.

З наукової точки зору, взаємодія з користувачем є ключовим елементом, що визначає, як користувачі взаємодіють із системою, а також які заходи можна впровадити для покращення цього процесу. Проектування бази даних, у свою чергу, передбачає організацію даних відповідно до предметної галузі, що дозволяє визначити, які саме дані слід зберігати і як різні елементи між собою взаємодіють.

З технічної точки зору, генерація коду є важливим процесом, що включає автоматичне створення коду для майбутнього використання на основі допоміжних елементів, які користувач може формувати у системі. Веброзробка забезпечує створення, побудову та підтримку вебсайту з використанням різних мов розмітки, що дозволяє впроваджувати код для розширення функціональних можливостей і покращення користувацького досвіду. Інженерія даних спрямована на розробку систем, які перетворюють необроблені дані у структуровану форму, наприклад, у реляційні таблиці або деревоподібні структури, що робить інформацію корисною для подальшої обробки та аналізу.

Таким чином, інтеграція наукових підходів до взаємодії користувача та проектування даних із технічними методами генерації коду, веброзробки та інженерії даних створює комплексну основу для розробки ефективної, зручної та продуктивної системи.

2.1 Нефункціональні вимоги до інтерфейсу та системи

До основних нефункціональних вимог, що визначають якість розробленої системи та її інтерфейсу, належать доступність та зручність використання, продуктивність і швидкодія, надійність та стійкість до

помилок, безпека, якість та валідність результатів, адаптивність і масштабованість, а також документованість і підтримка користувачів.

Велику увагу необхідно приділити інтерфейсу та роботі з таблицями, їх оформленню, модифікації та вказівкам різних атрибутів. Окремо необхідно виділити логіку побудови зв'язків між таблицями. Програма має бути простою та інтуїтивно зрозумілою у використанні, з чіткими інструкціями для кожної можливої дії, щоб уникнути непорозуміння з боку користувача.

Основний інтерфейс повинен мати окремий виділений простір для побудови таблиць, який має бути легко навігованим. Візуал має бути чітко структурований, мати інтуїтивно зрозумілі індикатори у вигляді назв, заголовків, додаткових коментарів і приємний дизайн , що виділяє ключові елементи, які потребують уваги та мінімізує час на навчання користувача роботі з системою. Особлива увага приділяється простоті виконання основних дій – створення, редагування та видалення таблиць, побудова зв’язків між ними, генерація SQL-коду. Використання досить коротких, але докладних назв кнопок, ясних прикладів, таких як малюнки, таблиці, діаграми, значно підвищить читабельність проекту, зробивши його доступнішим для кінцевої аудиторії [10].

Продуктивність і швидкодія системи мають забезпечувати миттєву обробку запитів користувачів, швидку генерацію ER-діаграм і SQL-коду, а також оперативний пошук у бібліотеці підсхем. Всі операції повинні виконуватися без затримок навіть при роботі з великими або складними схемами та запитами на природній мові. Для досягнення бажаної продуктивності необхідна оптимізація алгоритмів обробки та ефективне використання ресурсів системи.

Необхідно подбати про виключення будь-якого ризику збою чи помилки. Для цього програма повинна мати універсальні рішення для кожної проблеми та водночас підтримувати контрольований потік даних, щоб не виникали нестандартні випадки, а критичні значення також успішно

оброблялися. Надійність системи передбачає стійкість до збоїв та виключення ризику втрати даних або некоректної роботи у випадку виникнення непередбачуваних ситуацій. Впровадження механізмів перевірки коректності введених даних, обмеження дій користувача, фільтрації некоректних запитів, а також забезпечення відновлення роботи після збоїв є обов'язковими.

Безпека системи є важливим аспектом, особливо при зберіганні та передачі даних у хмарних сервісах, що використовуються для автентифікацію та збереження діаграм. Необхідно організувати клієнтсько- базову частину, зберігання діаграм користувачем, реалізувати автентифікацію користувачів, обмеження доступу до приватних схем, а також захист від несанкціонованих дій і витоку інформації.

Якість і валідність результатів гарантують коректність створених ER-діаграм та SQL-коду. Автоматична генерація коду має враховувати правила нормалізації, визначати первинні та зовнішні ключі, уникати надмірних зв'язків і забезпечувати сумісність отриманих структур із різними системами керування базами даних. Оцінка якості базується на критеріях коректності, оптимальності та практичності використання.

Адаптивність і масштабованість системи забезпечують її здатність працювати у різних сценаріях, підтримувати розширення функціоналу, інтеграцію нових моделей штучного інтелекту, додавання нових типів СКБД і бібліотек підсхем. Архітектура повинна підтримувати масштабування для роботи з великою кількістю користувачів і схем без значних змін у базовій структурі [11].

Документованість і підтримка користувачів передбачають надання чітких інструкцій, довідкових матеріалів та підказок, які будуть зрозумілі навіть користувачам із базовими знаннями у сфері баз даних. Особлива увага приділяється чіткій організації всіх розділів коду, щоб проект був чистим та зрозумілим для майбутніх інновацій. Це має бути зроблено шляхом структурування файлів та коментування коду. Ці нефункціональні

характеристики відповідають сучасним стандартам доступності, юзабіліті та безпеки, зокрема рекомендаціям WCAG, EN 301 549 та іншим міжнародним практикам, що забезпечують інклюзивність, надійність і комфорт користувачів з різними потребами [12].

2.2 Функціональні вимоги до системи

Фінальна робота з готовим функціоналом, передбачає дотримання вимоги щодо постійної перевірки даних, обмеження та фільтрації дій користувачів. Проект повинен мати набір правил, чітко визначаючи діапазон можливостей користувача. Цей набір правил дозволить як уникнути витоку скриптів з неіснуючими функціями, так і встановити конкретні межі, зрозумілі кінцевому користувачеві.

Важливою вимогою є також забезпечення архітектури клієнтської бази даних, яка дозволяє користувачеві зберігати і відновлювати роботу. Така архітектура забезпечує збереження роботи користувача і безперервність процесу проектування. Відповідність логіці дії-відповіді, яка буде передбачати максимальну близькість користувача до продукту через інтерактивність та реакцію проєкту. Кожна дія має мати передбачувані наслідки, а зміни повинні фіксуватися для подальшого контролю та відновлення.

Створення таблиць та зв’язків між ними є основним функціоналом системи. Інструмент повинен забезпечувати повноцінне керування таблицями, їх властивостями та зв’язками, включаючи можливість додавання атрибутів, визначення їх типів, встановлення обмежень, таких як NOT NULL, створення ключів та унікальних груп, а також видалення цих елементів. Така функціональність забезпечує користувачу зручний та інтуїтивний досвід роботи з БД. Автоматична генерація SQL-коду є важливою складовою проєкту. Система повинна синхронно оновлювати SQL-код відповідно до змін у таблицях, а також підтримувати генерацію

коду на кількох мовах запитів. Це забезпечить користувачам широкий вибір та гнучкість у роботі з різними платформами та СКБД. Проєкт повинен мати окремий код для конвертації таблиць в SQL-код з побудовою повноцінних таблиць на трьох різних мовах (MySQL, MSSQL, PostgreSQL).

2.3 Вибір технології та опис інструментів

Основним інструментом для реалізації проекту став JavaScript, який значно розширив функціональні можливості вебсторінки та дозволив створити власний функціонал. Завдяки JavaScript було реалізовано всі ключові рішення, використовуючи як базові можливості мови, так і спеціалізовані бібліотеки та API [13].

Для формування базової структури інтерфейсу застосовувалися HTML та CSS, які забезпечили створення зрозумілого та інтуїтивно доступного дизайну з чіткими кнопками та таблицями.

Важливим стало використання бібліотеки JsPlumb, яка надала можливість будувати зв’язки між об’єктами, зберігаючи при цьому структуру DOM, що було важливим для візуалізації ER-діаграм.

Для організації збереження даних і управління користувачами використовувалася платформа Google Firebase.

Оптимізацію зберігання даних було досягнуто за допомогою бібліотеки Pako, яка дозволяє стискати дані приблизно у чотири рази, що допомагає економити місце в базі даних і підвищувати ефективність роботи з великими обсягами інформації.

Для інтеграції з чат-ботом GPT використовувалося спеціальне API, що забезпечує коректну передачу запитів і отримання відповідей, що дозволяє автоматизувати процес генерації коду та інших функцій системи.

3 АРХІТЕКТУРА СИСТЕМИ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Розробка архітектури системи

У рамках роботи передбачено проєктування архітектури вебінструменту, який інтегрує модулі генерації, бібліотеку підсхем, графічний редактор ER-діаграм, а також інтерфейс для роботи із SQL- кодом. Такий підхід дозволяє створити єдину платформу, що охоплює всі основні етапи моделювання та реалізації РБД.

Система повинна забезпечувати можливість створення та редагування ER-діаграм, а також автоматичної генерації SQL-коду для різних систем керування базами даних, зокрема MSSQL, PostgreSQL та MySQL.

Окрім цього, важливим функціоналом є збереження, відновлення та повторне використання як схем, так і SQL-скриптів, що сприяє підвищенню ефективності роботи користувачів і забезпечує гнучкість у процесі проєктування.

Особливу увагу приділено проєктуванню бібліотеки підсхем, яка повинна підтримувати пошук і фільтрацію БД за ключовими словами, категоріями та іншими критеріями на основі природної мови. Додатково передбачається впровадження семантичного опису та класифікації структур баз даних, що дозволяє підвищити рівень автоматизації та інтелектуальної підтримки користувача під час роботи з великими обсягами схем і даних.

Запропонована архітектура візуального генератору схем для реляційних баз даних. Система працює з двома моделями штучного інтелекту: M_1 – генератор нової схеми за описом на природній мові, та M_2 працює як класифікатор схем за предметними галузями, реалізує пошук бібліотечної схеми за описом на природній мові (рисунок 3.1).

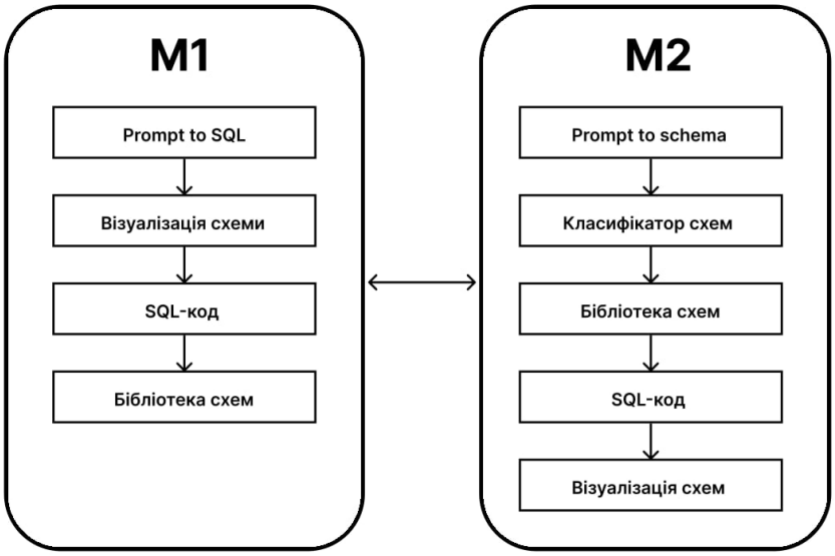

Рисунок 3.1 – Архітектура візуального генератору схем РБД

Модель M_1 може генерувати не лише SQL-код для трьох відомих СКБД (MSSQL, Postgre, MySQL), але й проводить його оптимізацію, що включає автоматичне додавання індексів для підвищення продуктивності запитів, використання оптимальних типів даних та усунення надлишкових зв'язків, що зменшує складність схеми. На рисунках 3.2 та 3.3 зображено інтерфейс додатку, де користувач може створити нову підсхему або знайти існуючу.

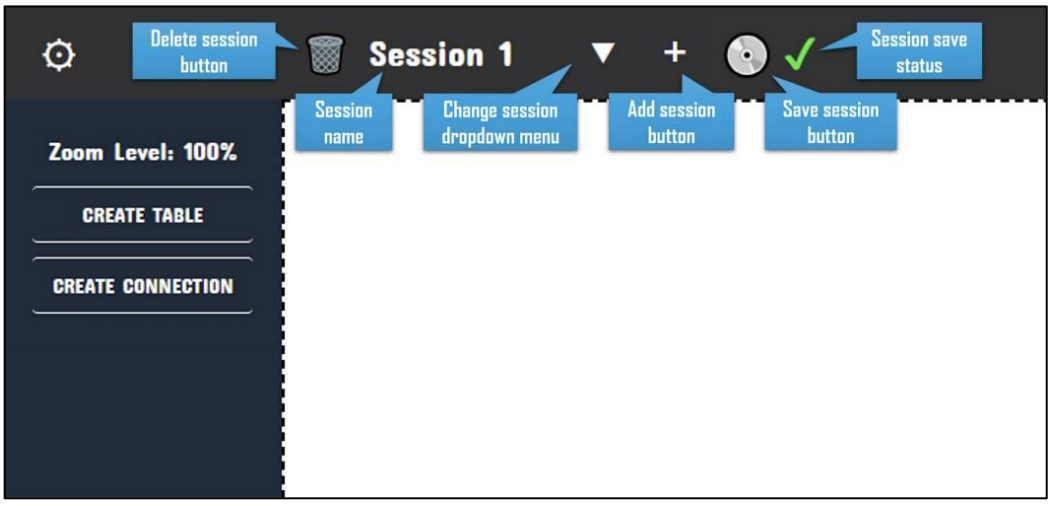

Рисунок 3.2 – Інтерфейс генератора бібліотечних схем, модель M_1

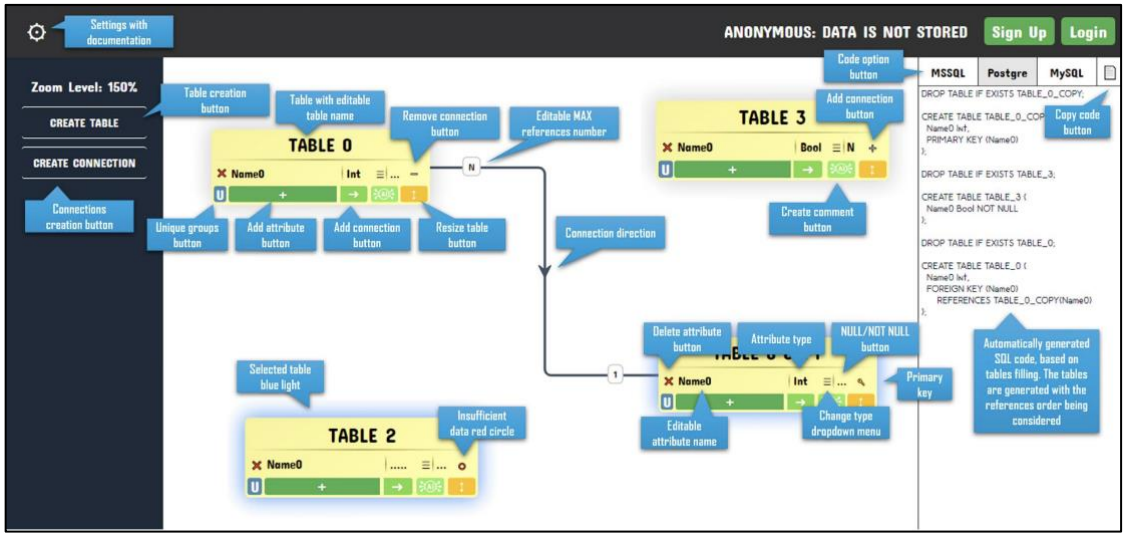

Рисунок 3.3 – Інтерфейс генератора бібліотечних схем, модель M1

3.2 Клієнт та база даних

Для виконання вимоги до архітектури бази даних клієнта необхідно створити модель, яка зберігає унікальний ключ користувача, отриманий під час автентифікації, зберігаючи дані у форматі дерева. Кожен користувач повинен мати ряд властивостей: останній вхід до системи, пошта, пароль, останній сеанс і всі сеанси, де кожен повинен мати таблиці та зв'язки між ними.

Ця модель повинна мати безпеку на основі правил, щоб зберігати дані у безпеці. Користувач, як клієнт бази даних, повинен мати можливість надсилати запити, пов'язані із застосованими змінами, та отримувати дані. Можливість фіксації змін має бути реалізована через ряд кнопок та пов'язаних функцій, які безпосередньо виконують дії створення нового сеансу, видалення існуючого сеансу, збереження змін сеансу.

Процес отримання даних із бази даних має бути реалізований у двох сценаріях: користувач вибирає один із сеансів або користувач перезавантажує сторінку.

3.3 Програмна реалізація

Для реалізації базової структури вебпроєкту були використані мови розмітки HTML та стилізації CSS. З використанням макетів та стилів, вони дозволили створити простий для розуміння інтерфейс із візуально зрозумілими кнопками та таблицями. Завдяки застосуванню цих технологій вдалося розділити вебсторінку на функціональні блоки, кожен із яких відповідає за виконання певних запитів і забезпечує інтуїтивну навігацію для користувача.

Кожне з наведених нижче рішень було реалізовано шляхом роботи з JavaScript, який дозволив впровадити складний функціонал і підвищити інтерактивність вебсторінки та спеціальними бібліотеками.

Зокрема, бібліотека JsPlumb забезпечила можливість створення візуальних зв’язків між об’єктами, зберігаючи при цьому цілісність структури DOM, що є важливим для коректної роботи графічних елементів.

Для оптимізації зберігання даних застосовувалась бібліотека Pako, яка дозволила суттєво зменшити розмір інформації, необхідний для збереження у базі даних.

Інтеграція з чат-ботом GPT здійснювалася через спеціальні API, що дозволяло надсилати коректно сформовані запити та отримувати якісні відповіді.

Для роботи з базою даних була використана платформа Google Firebase, яка дозволяє працювати з базою даних через код JavaScript. Цей інструмент не лише забезпечує збереження та вилучення інформації, а й дозволяє встановлювати рівні безпеки, створювати приховані функції та розміщувати вебдодаток на власному домені, що значно підвищує надійність системи. Завдяки Firebase можна легко налаштувати синхронізацію даних у реальному часі, що особливо корисно для динамічних додатків. Крім того, платформа підтримує зручні інструменти для аутентифікації користувачів, що спрощує процес реєстрації та входу. А

можливість інтеграції з іншими сервісами Google робить її ще більш універсальною та ефективною для розробки сучасних проєктів.

3.3.1 Безпечний інтерфейс. Створення таблиці

Можливість створення таблиці реалізована за допомогою кнопки «Створити таблицю», яка активує прослуховувач подій, щоб користувач міг натиснути та створити таблицю в полі діаграми.

Декілька методів були використані для того, щоб гарантувати, що таблиця випадково не буде створена за полем або заблокована від доступу до інших таблиць:

– event.target.closest («.object») – перевірка, чи створена таблиця поверх іншої таблиці, що забезпечується простою перевіркою при активації прослуховувача подій;

– setTableCounter () – генерація порядкового номера для таблиці, щоб їх імена залишалися унікальними;

– setSpawnPosition () – перевірка, чи знаходиться таблиця за межами кордонів при створенні, та автоматичне переміщення її всередину;

– checkUniqueness ( ) – спеціальна функція резервного копіювання для перевірки унікальності імен та у противному випадку їх заміни на будь-яке ім’я. Використовується як ознака незаповненої таблиці.

3.3.2 Функціональний інтерфейс таблиці

Після створення кожна таблиця активує ряд прослуховувачів подій, надаючи користувачеві такі можливості, призначені для побудови максимально функціонального та корисного інтерфейсу для роботи з діаграмами ER.

activEditMode () – можливість зміни імені таблиці, обмежена довжиною символів getTextWidth () та унікальністю нового імені checkUniqueness (). Відкат змін відбувається у разі порушення.

activResizeButton () – кнопка зміни розміру таблиці осі Y до одного видимого атрибута. Це враховує уникнення спроб збільшення таблиці внизу діаграми з допомогою функції resize ().

activeCommentButton () – далі активується можливість генерації тригерів на основі коментаря, але вона спеціально перевіряє обліковий запис, тому що для цього потрібен особистий ключ API користувача, який вводиться лише при вході в систему.

activConnectTableButton () – далі є можливість витягнути зв'язок із таблиці з ключовими атрибутами інших таблиць за допомогою автоматичної генерації нового заповненого атрибута у вихідній таблиці.

activAddButton () – далі надається можливість створення атрибутів усередині таблиці, але тільки якщо заповнено ім'я таблиці, інакше це призводить до створення цілої структури таблиць без генерації коду.

activUniqueButton () – можливість створювати унікальні групи атрибутів у таблиці, що також забезпечується функцією validation (), яка визначає чи має створена група сенс.

activObject () – в кінці об'єкт отримує можливість переміщення шляхом реалізації функції startDragging (). Точний та оптимізований код для перевірки меж таблиці приведено на рисунку 3.4.

Listing 1: Dragging table coordinates recalculation

```
var clampedX =
  Math.min(
    Math.max(newX, minX),
    maxX
  );
var clampedY =
  Math.min(
    Math.max(newY, minY),
    maxY
);
```

Рисунок 3.4 – Перерахунок координат таблиці під час перетягування

3.3.3 Функціональний інтерфейс атрибута

При додаванні атрибута до таблиці за допомогою відповідної кнопки з’являється операційний атрибут, який дозволяє виконувати різні дії як усередині таблиці, так і за її межами, забезпечуючи гнучке керування даними. Нижче наведено детальний опис функціональності кнопок та їхніх особливостей, із врахуванням безпечних обмежень, щоб користувач мав чітке уявлення про можливості інтерфейсу.

activDeleteButton () – кнопка видалення забезпечує безпечне видалення атрибута з таблиці з видаленням унікальних груп, до яких він належав та зв'язків.

activEditMode () – дозволяє змінити ім'я атрибута, обмежено getTextWidth () та checkUniqueness (). За порушення умов відбувається відкат змін.

activTypeButton () – кнопка створює невелике вікно у таблиці для вибору типу атрибута. Маючи кілька обмежень перед активацією неможливо встановити тип, якщо ім'я атрибута будь-яке, і неможливо змінити тип, якщо атрибут є зовнішнім ключем.

activNotNullButton () – дозволяє призначити властивість NOT NULL.

activConnectButton () – це друга кнопка, яка дозволяє з'єднувати таблиці один з одним, при цьому вихідний атрибут вже вибраний.

Відображається символ «+», тільки коли атрибут заповнено ім'я та тип. Відображається символ «-», коли атрибут пов'язаний лише з атрибутом із символом ключа. Відображається символ «ключ», коли «+» натиснути правою кнопкою миші, щоб встановити атрибут як ключ таблиці.

Повністю описаний інтерфейс таблиці та атрибутів, разом із зазначенням їх функціональності та безпечних обмежень на дії, дозволяє користувачеві побачити наступну картину (рисунок 3.5).

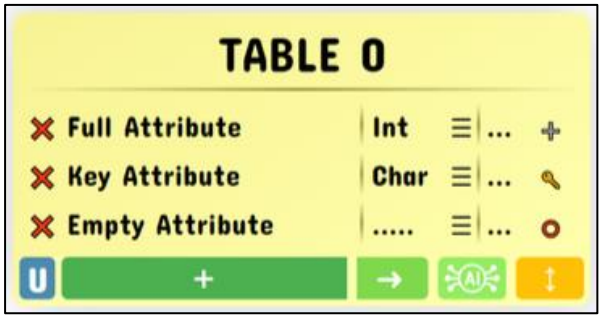

Рисунок 3.5 – Інтерфейс таблиці

3.3.4 Інтерфейс з'єднання таблиць

Окрім двох раніше описаних кнопок для створення зв’язків, у системі передбачена третя кнопка — «Створити з'єднання» поза межами діаграми. Ця кнопка має унікальну функцію: вона дозволяє встановлювати зв’язок типу «багато-до-багатьох» між двома вибраними таблицями, що значно розширює можливості моделювання складних структур даних.

Завдяки використанню бібліотеки JsPlumb вдалося реалізувати зручний і наочний вигляд з’єднань. Кожне з’єднання відображається з чітко визначеним напрямком і має дві мітки, які вказують на тип зв’язку. Одна з міток є редагованою і може приймати числові значення в діапазоні від 1 до 999 999, що дозволяє гнучко налаштовувати параметри зв’язку. Подвійне натискання на з’єднання відкриває детальну інформацію про нього, включаючи можливість видалення зв’язку, якщо це необхідно, тим самим забезпечуюч користувачу повний контроль над структурою зв’язків.

Інтерфейс таблиць доповнюється продуманим і функціональним інтерфейсом управління з’єднаннями. Це дозволяє завершити першу частину проектування бази даних, формуючи цілісну та зручну для роботи систему. Результат цього етапу, включно з візуалізацією з’єднань, представлено на рисунку 3.6.

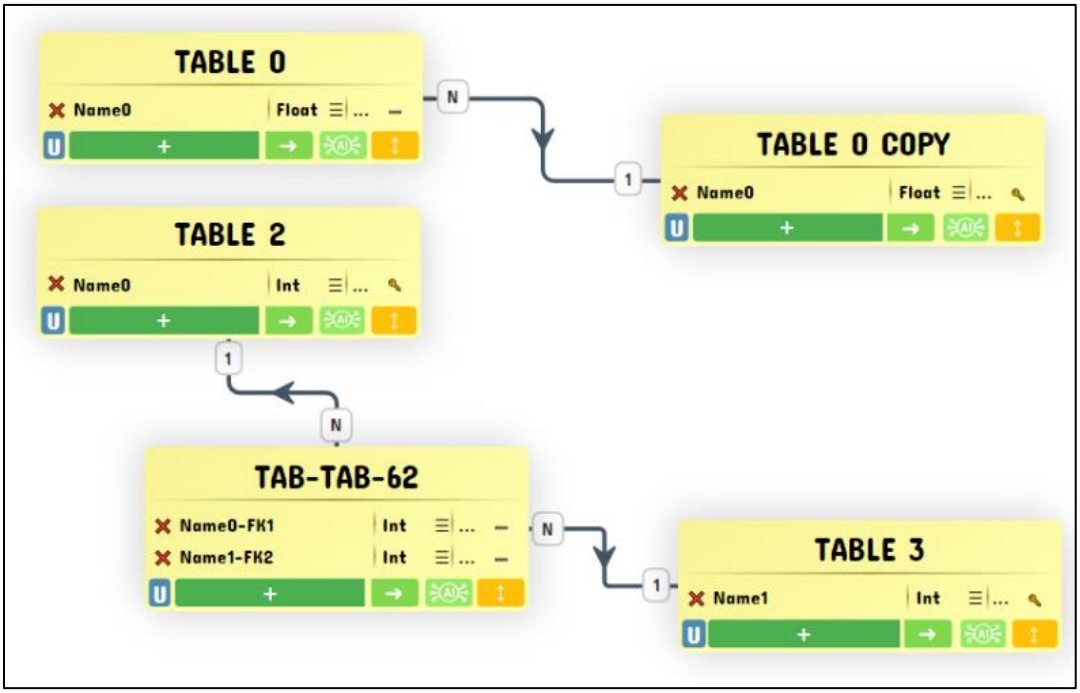

Рисунок 3.6 – З'єднання таблиць

3.3.5 Адаптивна логіка для додавання та видалення з'єднань

Адаптивна логіка поєднує виконання вимог логіки дії-відповіді та створення таблиць та зв'язків. Інтерфейс робить це не тільки більшою варіативністю безпечних дій, але і адаптивним характером. Таким чином, різні дії користувача можуть впливати, надаючи додаткові можливі дії для виконання. Такий підхід дозволяє користувачеві не робити зайвих кроків, забезпечуючи безпечнішу та ефективнішу взаємодію.

Для забезпечення адаптивної логіки було застосовано низку рішень, що призвело до створення програми на основі функцій і активаторів, що викликаються по порядку, де кожний може спричинити виклик проміжних функцій.

Ця логіка може бути частково описана у вигляді графової системи, де певні дії спричиняють інші дії, що відповідає дизайну (рисунок 3.7). Ця структура графа демонструє, як певні кнопки та ярлики призводять до додавання зв'язків з таблиці двома способами, один з яких генерує новий

атрибут у таблиці. Кожен вузол у графі може бути видалений або видалити інший вузол. Наприклад, можна видалити лише зв'язок, зберігши рядок.

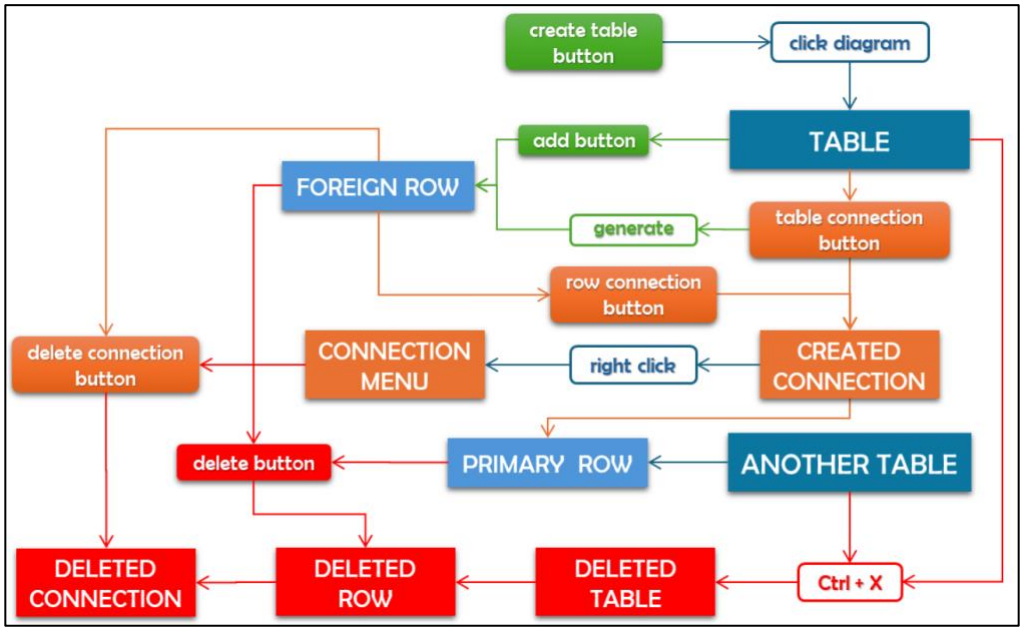

Рисунок 3.7 – Адаптивна логіка для додавання та видалення з'єднань

3.3.6 Розширені механізми адаптивної логіки

Щоб глибше вивчити адаптивну логіку та надати кілька її прикладів, розглянемо структуру графа наступних дій (рисунок 3.8).

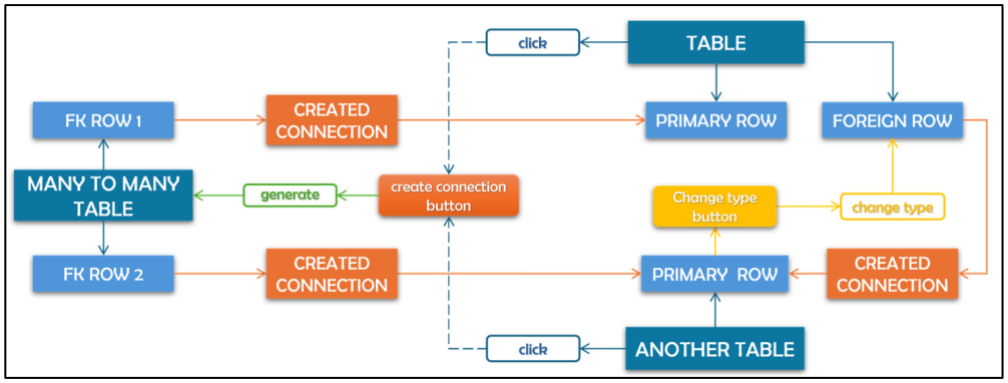

Рисунок 3.8 – Адаптивна логіка для створення з'єднань «багато до багатьох» та зміни типу зовнішнього ключа на основі первинного ключа

Структура, з одного боку, описує процес створення третього зв'язку для двох первинних ключів з генерацією проміжної таблиці з логікою «багато-до-багатьох».

Спочатку використовується функція createObjectBetween () для генерації таблиці, спеціально обраної для виконання відношення, щоб отримати додаткові відомості про реалізацію коду того, як обчислюються координати (рисунок 3.9).

Listing 2: Intermediate many-to-many connection table generation

```
createTableBtn.dispatchEvent(
  new Event('click')
);

[object1, object2] = selectedObjects;
var rect1 = object1.getBoundingClientRect();
var rect2 = object2.getBoundingClientRect();

midX = ((rect1.left + rect2.left) / 2)
       + 87 * scale_moving;
midY = ((rect1.top + rect2.top) / 2)
       + 28 * scale_moving;;

diagramContent.dispatchEvent(
  new MouseEvent('click',
    {
      clientX: midX,
      clientY: midY
    }
  )
);
```

Рисунок 3.9 – Генерація проміжної таблиці для зв'язку багато до багатьох

Потім для нової таблиці генеруються два зовнішні ключі за допомогою функції changeNameAddTwoRows (), кожен з яких приймає ім'я та тип первинних ключів, на які вони посилаються.

З іншого боку, описаний принцип успадкування типів даних, коли зміна типу первинного ключа призводить до автоматичної зміни типу зовнішнього ключа. Щоб побачити, як зміна типу ключа знаходить та змінює точний тип атрибута (рисунок 3.10).

Listing 3: Foreign key automatic type change based on referenced key

```
function getSources(FindRow,sources) {
  allConnections.forEach(connection => {
    if(connection.ToRow == FindRow){
      sources.push(connection.FromRow);
    }
  });
  return sources
}
...
rowSources = getSources(
  chosenTdType.parentElement,[]
);

rowSources.forEach(row => {
  row.querySelector("[class^='type-cell-']")
  .textContent = option;
});
```

Рисунок 3.10 – Автоматична зміна типу зовнішнього ключа на основі типу пов'язаного (референтного) ключа

Описана адаптивна логіка тісно пов'язана з безпекою інтерфейсу, вона дозволяє користувачеві уникати непотрібних дій в одних місцях і уникати помилкових в інших.

Що стосується структури бази даних, Google Firebase було обрано хмарним сховищем для проекту. Використання введеної пошти дозволяє згенерувати унікальний ключ користувача, який може бути переданий до бази даних із створенням гілки для конкретного облікового запису.

Структура сховища є деревом, що дозволяє легко призначати обмеження і каскадні правила, і кожен користувач може мати ряд сесій, до яких може отримати доступ тільки він сам після входу в систему та отримання унікального ключа користувача. Таким чином, база даних дозволяє повністю реалізувати безпечне зберігання даних та доступ користувача до них. На рисунку 3.11 показано, яку деревоподібну структуру має проект.

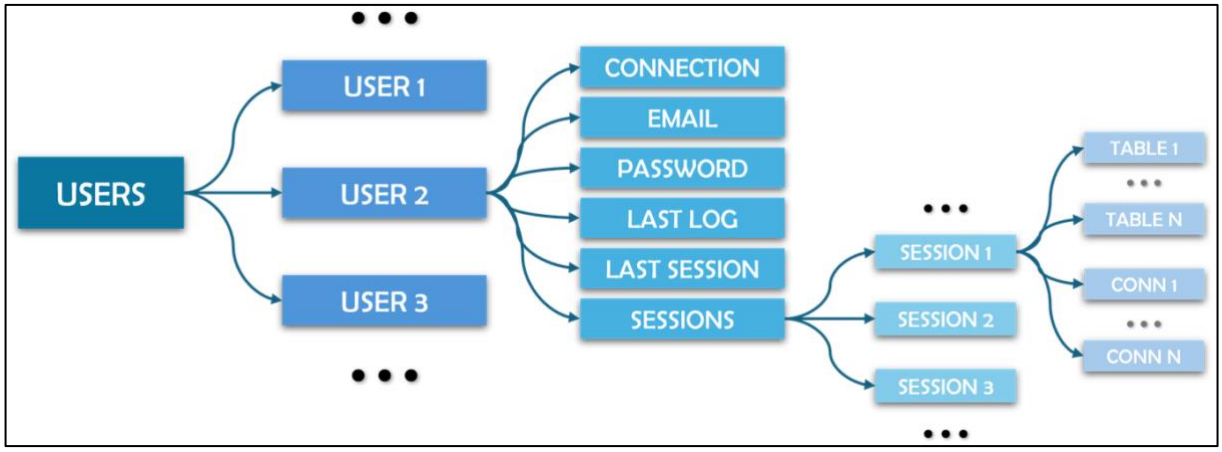

Рисунок 3.11 – Деревоподібна структура користувачів бази даних Firebase

На зображенні показано, що, згідно з проектом, кожен користувач має базовий набір властивостей. Вони забезпечують як безпечну взаємодію з системою, так і зручне керування обліковими записами.

Пошта та пароль є ключовими елементами для автентифікації користувача під час входу в систему. Вони виступають основним механізмом перевірки, гарантуючи безпеку доступу до облікового запису.

Останній вхід до системи – це властивість, яка дозволяє потенційному адміністратору побачити, як довго користувач перебуває у мережі програми.

Останній сеанс – забезпечує зручність для користувача та дозволяє користувачеві відразу перейти до роботи під час входу до системи або оновлення сторінки.

Підключення – статус, який запобігає численним входам в один і той же обліковий запис.

Сеанси – представляють собою набір сесій користувача, де кожна сесія структурована як сукупність таблиць і зв’язків. Це дозволяє чітко організувати дані, пов’язані з діяльністю користувача в системі, та забезпечує їх логічне групування.

Для певних гілок системи можуть застосовуватися спеціальні обмеження, які стосуються як доступу до даних, так і їхньої форми. Ці

обмеження детально описані в коді правил, представленому на рисунку 3.12, що забезпечує чітке регулювання взаємодії з системою.

Listing 4: Firebase database security rules

```
{"rules": {
  "users": {
    "$uid": {
      ".read": "$uid === auth.uid",
      ".write": "$uid === auth.uid",
      "email": {
        ".validate": "newData.isString() &&
                newData.val().length <= 320"
      },
      "lastSession": {
        ".validate": "newData.isString() &&
                newData.val().length <= 20"
      },
      "lastLog": {
        ".validate": "newData.isString() &&
                newData.val().length <= 30"
      },
      "password": {
        ".validate": "newData.isString() &&
                newData.val().length <= 30"
      },
      "connection": {
        ".validate": "newData.isBoolean()"
      },
      "sessions": {
        "$session": {".validate": true}
      },
      "$other": {".validate": false}
    }
  }
}}
```

Рисунок 3.12 – Правила безпеки бази даних Firebase

3.3.7 Збереження та вилучення клієнта

У додатку для можливості входу в обліковий запис, щоб забезпечити взаємодію користувача з базою даних, були підготовлені такі функції та можливості, як показано на рисунку 3.13.

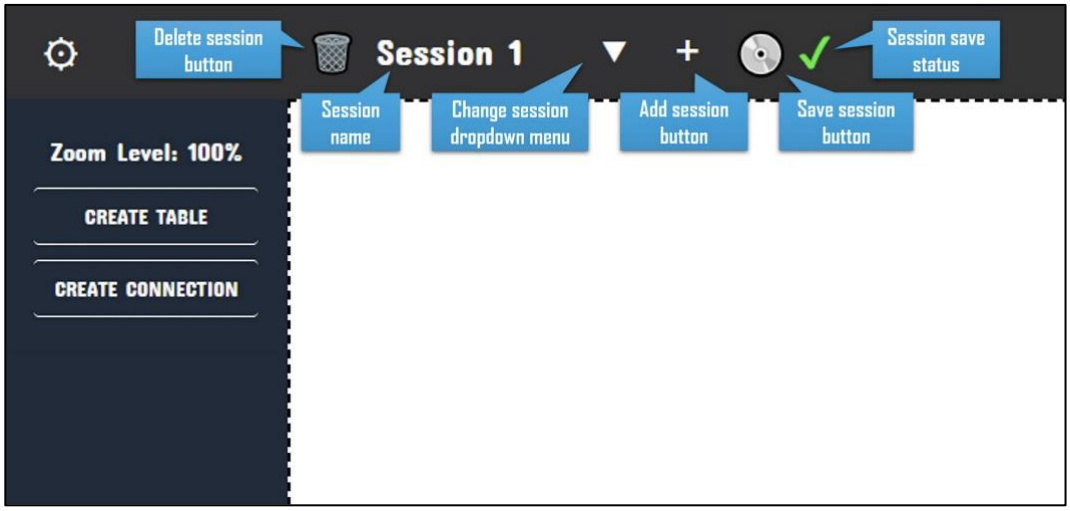

Рисунок 3.13 – Інтерфейс користувача веб-сайту для взаємодії з базою даних

Можна побачити, як організовано простір для користувача, щоб зберігати та отримувати дані. Кожна кнопка має функції, реалізовані за нею.

Видалення сеансу дозволяє користувачеві видалити поточний сеанс із можливістю автоматичного перемикання на існуючий сеанс. Ця функція реалізована за допомогою deleteSession (), який спочатку видаляє всю гілку сеансу, потім перезавантажує всі сеанси за допомогою getSessions () для оновлення списку та встановлює перший існуючий сеанс як останній сеанс. Наприкінці він використовує getLastSession () для отримання всієї діаграми.

Змінення сеансу при використанні відображає меню, що розкривається, за допомогою formDropdownMenuSession (), яке показує параметри сеансу, вибравши один з яких користувач активує saveSession () для збереження результатів поточної таблиці, і призначає новий як останній і активує getLastSession ().

Додавання сеансу можливе використанням кнопки, яка генерує модальне вікно з addSessionWindow (), де користувач може ввести ім'я сеансу з можливістю введення короткого опису. Створений сеанс генерує нову гілку, активується функція getSessions () для оновлення списку сеансів,

водночас функція getLastSession () призначає створений сеанс останньому, очищаючи діаграму від попереднього сеансу, якщо він існував.

Зберігання сеансу реалізує збереження всієї схеми з перевіркою на порожні поля для її свідомості. Основна функція – saveSession (), яка використовує згадану бібліотеку pako для стиснення кожної таблиці. Після того, як словник таблиць і з'єднань згенерований, він зберігається в базі даних. Короткий фрагмент (рисунок 3.14) пропонується для того, щоб зробити деталі зрозумілішими.

Listing 5: Code implementation for saving sessions to database

```
return new Promise((resolve, reject) => {
  ... check section ...
  ...
  const updates = {};
  onValue(dbRefCreation, (snapshot) => {
    const creation = snapshot.val();
    remove(dbRef).then(() => {
      updates["creation"] = creation;

      ....forEach(object => {
        var id = object.getAttribute("id");
        ... object pako compression ...
        updates[id] = compressedData;
      });

      var connNumber = 0;
      allConnections.forEach(conn => {
        ... assign connection vars ...

        updates["conn-" + connNumber++] =
        fromId + " " + fromRow + " " + toId
        + " " + toRow + " " + fromLab;
      });

      update(dbRef, updates).then(() => {
        changeSaveStatusSession("close");
        resolve();
      }).catch(error => { reject(error); });
    })
  },{ onlyOnce: true });
});
```

Рисунок 3.14 – Реалізація коду для збереження сесій у базу даних

Роботу коду також можна описати як спочатку збирання всіх даних, необхідних для збереження, і тільки потім їх передачу в базу даних одним

запитом. Такий підхід надає користувачеві необхідні дії для керування обліковим записом.

3.3.8 Логічний граф генерації SQL-коду, заснований на умовах та діях

Для забезпечення ефективної автоматичної генерації SQL-коду необхідно впровадити механізми, що синхронізують зміни в структурі таблиць із відповідними модифікаціями коду. Ключовим елементом є динамічне відображення логіки ER-діаграм у SQL-запитах, яке забезпечує консистентність (узгодженість або цілісність даних) між візуальним представленням схеми та її програмною реалізацією.

Система повинна підтримувати генерацію коду для різних СКБД (наприклад, MySQL, PostgreSQL, MSSQL), враховуючи специфіку їхніх діалектів SQL. Для цього використовуються шаблони трансформації онтологічних моделей, де кожна мова має унікальний набір правил для визначення типів даних, обмежень та індексів. Наприклад, автоматичне перетворення атрибутів ER-діаграми у DDL-інструкції (CREATE TABLE, ALTER TABLE) реалізується через синтаксичні дерева, що адаптуються під обраний діалект.

Для спрощення процесу проектування реалізовано функцію створення повної схеми БД на основі текстового опису. Система аналізує вхідні дані, ідентифікує сутності, атрибути та зв’язки, після чого формує ER-діаграму та відповідний SQL-код. Це забезпечує зменшення часу розробки та мінімізацію помилок, пов’язаних із ручним введенням.

Синхронізація змін досягається завдяки подійно-орієнтованій архітектурі, будь-яка модифікація таблиці (додавання стовпця, зміна типу даних) ініціює перегенерацію відповідних ділянок коду. Використання інкрементних алгоритмів дозволяє оновлювати лише ті частини SQL- скриптів, які зачіпаються змінами, що підвищує продуктивність системи.

Таким чином, поєднання шаблонів трансформації, NLP та подійно-орієнтованого підходу забезпечує створення інструменту, здатного генерувати валідний SQL-код, адаптований під різні СКБД, із збереженням цілісності даних та мінімальними витратами часу на коригування.

Як було зазначено, додаток має адаптивну логіку. Основна відповідь на кожну дію користувача відображається у правій вкладці з кодом, який постійно оновлюється.

Принцип генерації коду заснований на аналізі таблиць, що мають конкретну структуру. Фактично все зводиться до переведення однієї структури (ER-діаграми) в іншу (SQL-код), але з урахуванням нових можливостей, які будуть описані далі. Реалізація генерації коду поділена на конкретні набори функцій. Дві основні функції SQlForAll () та createFullSQLTable (). Їх реалізація коду наведена на рисунку 3.15.

Listing 6: Two main SQL generation functions for whole schema construction

```
function SQlForAll() {
  ....forEach(object => {
    createFullSQLTable(object);
  });
  allConnections.forEach(conn => {
    createForeignKey(conn.FromObject,
      conn.FromRow,conn.ToObject,conn.ToRow);

    if(conn.FromLabel != "N") {
      createFKConstraint(conn.FromObject,
        conn.FromRow,conn.FromLabel);
    }
  });
}

function createFullSQLTable(object) {
  ... table name check ...
  object...forEach(row => {
    createSQLRow(object, row);
  });
  createUniqueGroups(object);
}
```

Рисунок 3.15 – Дві основні функції генерації SQL для побудови всієї схеми

Принцип побудови кожної таблиці з урахуванням коду у тому, щоб спочатку побудувати все таблиці, їх унікальні групи, та був зв'язок між ними. Таким чином зберігається чіткий порядок генерації коду.

Для спрощення процесу генерації таблиць у функції createFullSQLTable () використовується лише функція створення рядка createSQLRow (). Це можливо, оскільки генерація коду успадковує каскадну природу проекту, сама функція підтягує багато інших функцій. Цю тезу можна представити як граф алгоритму (рисунок 3.16).

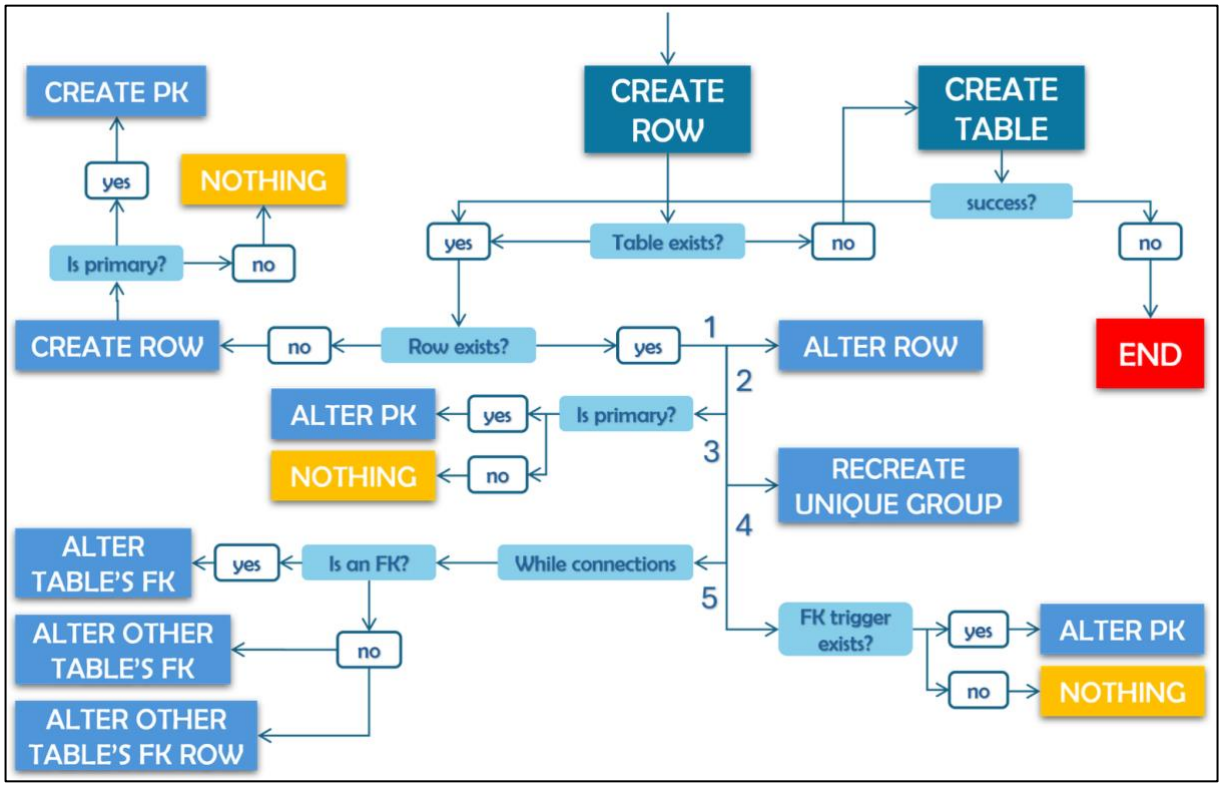

Рисунок 3.16 – Логічний граф генерації коду SQL, заснований на умовах та діях.

Граф повністю описує цю логіку, реалізуючи покрокову перевірку питань для створення за сценарієм. У певних сценаріях генерація активує необхідні функції, особливо у випадках, коли атрибут вже існує, оскільки це спричиняє оновлення у всіх компонентах, де з'являється ім'я атрибута. Після

створення таблиці вона доповнюється унікальними групами та зовнішніми ключами для отримання повного коду.

З описаного коду та діаграми видно, що кожна таблиця поділена на певні розділи: атрибути, первинний ключ, унікальні групи, зовнішні ключі (рисунок 3.17).

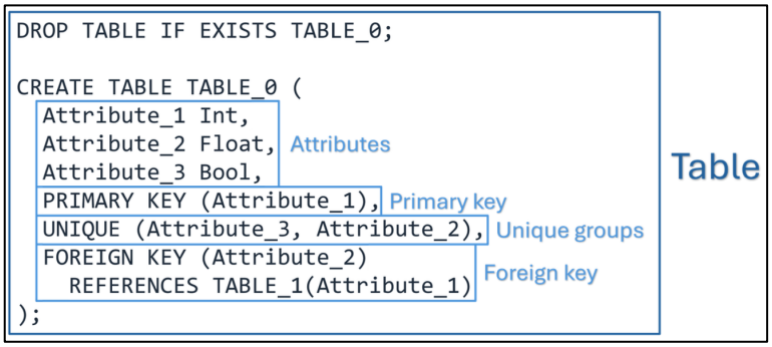

Рисунок 3.17 – Згенерована структура таблиці SQL

Як приклад їхньої генерації взято розділ атрибутів. Частина функції CREATE ROW розбирає атрибут окремі частини, після чого збирає їх разом.

Докладніше це можна побачити на рисунку 3.18.

Listing 7: Code implementation of attribute generation into SQL code

```
... get sqlTable ...
var row_text = getAttrCellContent(row);

var sqlRow = document.createElement("span");
sqlRow.setAttribute(
  "id",
  "sql-row-" + row.classList[0]
);
sqlRow.classList.add("sql-row");
sqlRow.textContent = "  " + row_text + " " +
  row.querySelector(
    "[class^='type-cell-']"
  ).textContent.trim()
  +
  (row.querySelector(
    "[class^='not-null-button-']"
  ).textContent == "N" ? " NOT NULL" : "");
```

Рисунок 3.18 – Реалізація коду для перетворення атрибутів у SQL-код

Ще одне складне завдання генерації такого коду за шаблоном – робота із комами та порядком самих розділів, оскільки це вимагає постійної перевірки наявності попередніх або наступних розділів. Принцип ком на основі постійної перевірки сусідніх елементів, це також впливає на видалення елементів таблиці з відтворенням ком.

Ці перевірки для додавання кожного розділу забезпечують чистий та легко редагований код. Його основна ефективність, як було сказано раніше, полягає у зміні лише тих частин, які необхідно змінити. Крім створення розділів, є також змінні функції, що реалізують просту реконструкцію на основі функцій створення та функцій видалення з повторною перевіркою ком після вирізання розділу. Підсумовуючи, всі три фрагменти коду з діаграмами, які додають контекст, дають повне пояснення того, як була виконана перша частина дизайну генерації коду, принцип, що лежить в основі його, і наскільки він ефективний у проекті (рисунок 3.19).

Listing 8: Comma spacing generation inside SQL tables for consistency

```
var rows = sqlTable.querySelectorAll(
  "[id^='sql-row-new-row-']"
);
var lastRow = rows[rows.length - 1];
var sqlTableEnd = sqlTable.querySelector(
  ".sql-table-last-row"
);
if(lastRow) {
  lastRow.textContent +=
    lastRow.textContent.slice(-1)==","?"":",";
    sqlTable.insertBefore(
      sqlRow,
      lastRow.nextSibling.nextSibling
    );
    sqlTable.insertBefore(
      document.createElement("br"),
      sqlRow.nextSibling
    );
    if(sqlRow.nextSibling.nextSibling
       != sqlTableEnd) {
        sqlRow.textContent += ",";
    }
} else {
  sqlTable.insertBefore(sqlRow,sqlTableEnd);
  sqlTable.insertBefore(
    document.createElement("br"),
    sqlTableEnd
  );
}
```

Рисунок 3.19 – Генерація відступів після ком у SQL-таблицях для узгодженості (єдиного стилю)

Інтеграція штучного інтелекту відіграє ключову роль у створенні складних елементів, таких як тригери або процедури. Алгоритми NLP (обробка природної мови) аналізують текстовий опис користувача, виділяють сутності, зв’язки та умови, після чого генерують відповідний код.

3.3.9 Генерація SQL-коду моделями ШІ

У даній роботі розглядається розробка двох моделей ШІ, призначених для підтримки процесів проєктування та роботи з БД. Ці моделі виконують різні, але взаємодоповнюючі функції, що сприяють підвищенню ефективності та зручності в процесі створення й управління базами даних.

Перша модель, позначена як M1, відповідає за автоматизоване створення нових схем баз даних та відповідного SQL-коду на основі описів, наданих природною мовою. Вона оснащена механізмами оптимізації, які дозволяють автоматично додавати індекси для покращення продуктивності запитів, обирати найбільш підходящі типи даних та усувати надлишкові зв’язки. Завдяки цьому структура бази даних стає менш складною, а її ефективність значно зростає.

Друга модель, M2, зосереджена на пошуку, фільтрації та семантичному описі готових бібліотечних схем за допомогою текстових запитів. Це дає змогу класифікувати існуючі рішення та повторно використовувати їх у нових проєктах. Застосування методів обробки природної мови та семантичного аналізу забезпечує інтелектуальний доступ до великої кількості структур баз даних, що значно полегшує процес вибору та адаптації схем для потреб конкретного проєкту.

Для забезпечення гнучкості в генерації коду, зокрема тригерів та цілісних схем, доцільно використовувати сучасні інструменти, які підтримують обробку запитів природною мовою. Це реалізується завдяки інтеграції через API-ключ, який активується під час входу користувача в обліковий запис. Програмна реалізація цього процесу наведена на

рисунку 3.20. Такий підхід дозволяє створювати зручні та ефективні рішення для проєктування баз даних, мінімізуючи ручну працю та підвищуючи якість кінцевого продукту.

Як можна бачити, відправлення запиту складається із трьох частин – метод, заголовки та тіло, сама функція повертає відповідь у форматі json , який аналізується для отримання необхідного коду.

Listing 9: Code to request chatGPT response based on prepared prompt

```
const requestData = {
  model: "gpt-3.5-turbo-0125",
  messages: [
    { role: "user", content: prompt }
  ],
};

fetch(
 'https://api.openai.com/v1/chat/completions',
{
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
    'Authorization': 'Bearer ${API_KEY}'
  },
  body: JSON.stringify(requestData)
})
.then(response => response.json())
.then(data => {
  GPT_OUTPUT =
    data.choices[0].message.content;
  resolve();
})
.catch(error => reject(error));
```

Рисунок 3.20 – Код для запиту відповіді ChatGPT на основі підготовленого запиту (prompt)

Кожна таблиця має кнопку для виклику модального вікна для створення коментарів до тригерів. Користувачеві необхідно ввести опис обмеження або дії, щоб отримати фрагмент коду та скопіювати його.

Шаблон запиту заснований на 3 вхідних параметрах: таблиця, коментар та мова.

Таблиця – додаток забезпечує генерацію тригерів тільки в таблицях, чого має бути достатньо для створення схеми.

Коментар – це вхідні дані ШІ для встановлення того, що має бути перевірено або виконано тригером у відповідь на деяку зміну таблиці.

Мова – як було сказано раніше, користувач має можливість працювати з трьома мовами, тому генерація коду через ШІ виконується трьома мовами.

Усі три параметри можна описати наступним шаблоном запиту, доповненим конкретними налаштуваннями для покращення результату (рисунок 3.21).

Listing 10: Trigger generation prompt pattern

```
const prompt = `
  Given SQL table:
    ${sqlTable}
  Generate SQL trigger in ${language}
  Of structure:
    ${structure}
  For given comment:
    ${comment}

  REQUIREMENTS:
    NO COMMENTS
    ONLY CODE
    PAY ATTENTION TO ATTRIBUTE TYPE
`;
```

Рисунок 3.21 – Шаблон запиту для генерації тригера

Потім шаблон відправляється в раніше описаний запит, щоб отримати результат і вивести його. При створенні сеансу, як описано раніше, користувач може ввести опис сеансу єдиний параметр, необхідний для генерації схеми. Після введення коментаря потім передається в наступний шаблон, який також доповнює його спеціальними налаштуваннями для кращої продуктивності (рисунок 3.22).

Listing 11: Schema generation prompt pattern

```
const prompt = '
  Generate SQL schema with tables STRICTLY of
  structure WHERE ONLY table and attributes
  are required:
    ... JSON PATTERN ...

  For given comment:
    ${comment}

  SCHEMA REQUIREMENTS:
    NO COMMENTS
    ONLY JSON FORMAT
    MINIMUM 3 TABLES
    MAXIMUM 16 TABLES

  TABLE REQUIREMENTS:
    ATTRIBUTES NAMES IN EACH TABLE ARE UNIQUE
    PRIMARY KEY IS ONLY ONE ATTRIBUTE
    NO RECURSIVE KEY REFERENCES
';
```

Рисунок 3.22 – Шаблон запиту для генерації схеми

Його принцип полягає в отриманні результату у форматі json , який був окремо написаний, представляючи структуру кожної таблиці у форматі можливих властивостей json. Для кращого розуміння наведено код на рисунку 3.23.

Listing 12: JSON-based prompt pattern for tables generation

```
[{
  "table" : (table name),
  "attributes" : [{
    "attribute_name" : (attribute name),
    "attribute_type" : (attribute type),
    "null" : (true/false)
  },...],
  "primaty_key" : (attribute name),
  "unique_groups" : [
    [(attribute_name), ...],
  ...],
  "foreign_keys" : [{
    "attribute_name" : (attribute name),
    "foreign_table" : (foreign table name),
    "foreign_attribute" : (foreign attr name),
    "max_references" :
      (number from 1 to 999999 | infinity "N")
  },...]
},...]
```

Рисунок 3.23 – Шаблон запиту на основі JSON для генерації таблиць

Потім цей висновок можна легко відтворити у форматі таблиці, використовуючи JSONtoSchema (). Він проходить по кожному елементу списку json, імітуючи на більшості кроків дії користувача, викликаючи функції для створення таблиць та заповнюючи їх по одній атрибутах, потім з'єднуючи всі таблиці, якщо є зв'язки. Суть у тому, що генерація коду на основі ШІ чудово доповнює роботу автоматичної генерації, додаючи різноманітність та більше можливостей для користувача, справляючись із завданням проектування.

З усього вищесказаного можна зрозуміти, що основне завдання динамічної генерації коду зі зручною побудовою збережених ER-діаграм добре виконується програмою. Весь розділ генерації коду є основним елементом, необхідним проведення дослідження того, наскільки позитивно виділений інструмент впливає ефективність і швидкість процесу проектування бази даних.

3.4 Тестування

Експериментальне тестування системи із залученням користувачів для оцінки швидкості створення ER-діаграм і SQL-запитів, точності коду, зручності інтерфейсу та валідності згенерованих структур дуже важливий етап цієї роботи. Реалізація та рішення докладно документовані, включаючи обговорення властивостей проекту. Надано всебічне пояснення інструменту побудови ER -діаграм. Велика увага приділяється методології генерації коду та її ефективності.

Перший етап тестування присвячений вивченню різних критеріїв з метою оцінки переваг використання розробленого інтерфейсу.

Наступним кроком є підготовка експериментів для користувачів, що сприятимуть отриманню неупереджених та об’єктивних оцінок за визначеними критеріями.

Завершальним етапом передбачено проведення серії експериментів із залученням користувачів, під час яких збираються їхні оцінки та коментарі.

На основі отриманих даних формуються результуючі бали, що дозволяють оцінити релевантність і ефективність розробленого інструменту та дати відповідь на поставлене наукове питання. Саме дослідження має практичне значення для галузі баз даних та галузі побудови діаграм, що використовуються для проєктування структури даних у проекті. Очікується, що результати роботи відповідатимуть як функціональним, так і нефункціональним вимогам, забезпечуючи комплексний підхід до оцінки та вдосконалення розробленої системи.

Тестування розділено на три частини, що складаються зі створення критеріїв оцінки, підготовки відповідних завдань щодо розкриття критеріїв та проведення оцінки результатів, отриманих у ході експериментів.

Критерії тестування повинні стосуватися як роботи з побудови діаграми, так і якості отриманої моделі, таким чином, фокусуючись на користі інструменту якості. Завдання повинні бути помірно складними, забезпечувати можливість використовувати будь-які інструменти для виконання завдань, ґрунтуватися на критеріях, пов'язаних із побудовою, виявляти здібності користувача у питанні швидкості та кількості відтворення таблиць для діаграми та створення коду на її основі. Завдання мають бути зосереджені на якості діаграм та фрагментів коду, щоб побачити, наскільки добре користувач зміг написати правильний код.

Для написання SQL-коду потрібен час на перенесення всього з діаграми в код і ще більше, коли необхідно реалізувати тригери з перевірками. Час для побудови ER-діаграми – найбільш базовий параметр при оцінці створення діаграми і тестування спрямовано на вивчення того, чи може інструмент дозволити користувачеві заощадити час і поліпшити розуміння діаграми.

Залежно від розміру моделі час побудови може змінюватись, але користувач все одно витрачає час на візуалізацію діаграми та розуміння її

взаємозв'язків. Тест налаштований так, щоб показати, автоматична генерація коду може значно заощадити час користувача, уникаючи необхідності будь-яких попередніх знань щодо написання SQL-коду. Увага звертається на питання часу переписування коду з одного до іншого.

Тривале використання – критерій визначення того, чи дозволяє інтерфейс пізніше повернутися до роботи, заощаджуючи час на реконструкцію коду і зміна структури бази даних.

Схема валідності – важливий фактор, який відноситься до оцінки якості створених діаграм, їх змісту, можливості взаємодії з ними та їхньої загальної коректності. Валідність SQL – цей критерій впливає якість написаного SQL- коду, його коректність в логіці самої схеми і можливість запуску коду.

Критерій безпеки – звертає увагу на те, чи надає інтерфейс можливість запобігти помилкам користувачів або пропуску деталей, дозволяючи їм рухатися у певному напрямку в інтерфейсі, не гублячись.

Крім цього для тестування необхідно скласти набір правил, з урахуванням яких можна оцінити фактичну корисність інтерфейсу, наданого проектом. Ці критерії повинні бути виключно об'єктивними, що мають на увазі певні одиниці, які будуть вимірюватися досвідом користувача. Очікується, що ці правила працюватимуть у форматі порівняння з оцінками тривіального кодування на ER-діаграмі або побудови її з нуля.

Для складених правил необхідно створити завдання. Вони повинні складатися з декількох вправ, спрямованих на те, щоб спонукати користувача використовувати SQL-код і будувати ER діаграми з урахуванням своїх можливостей і знань. Такі завдання повинні давати повну оцінку відповідно до умов.

Отримані результати після виконання завдань, були відображені, показуючи зміну даних критеріїв з використанням і без використання інструменту побудови ER. Також було проведено докладний аналіз, щоб відповісти на питання, наскільки позитивний вплив зробив інструмент на

ефективність створення бази даних на основі того, як змінилися критерії без використання та використання інструменту.

Оцінка продуктивності коду показала, що він дає швидкі та успішні результати, працюючи динамічно, тим самим витрачаючи мінімально можливий час отримання бажаного результату (рисунки 3.24 – 3.26).

User	DUI (min)	Approach (min)
1	11	30
2	9	24
3	9	29
4	8	29
5	10	27
6	8	26
7	10	25
8	11	28
Average advantage: 2.9 times		

Рисунок 3.24 – Порівняння часу між спеціалізованим інтерфейсом користувача та підходами самих користувачів

User	DUI (score)	Approach (score)
1	9.0	6.0
2	9.5	3.5
3	9.0	5.0
4	9.0	4
5	8.5	5.5
6	9.5	5.0
7	9.0	6.0
8	9.5	4.5
Average advantage: 1.9 times		

Рисунок 3.25 – Порівняння безпеки та коректності схеми між спеціалізованим інтерфейсом користувача та підходами самих користувачів

User	DUI (mistakes)	Approach (mistakes)
1	0	4
2	0	4
3	1	3
4	0	3
5	1	3
6	1	2
7	0	4
8	0	3
Average advantage: 3 mistakes less		

Рисунок 3.26 – Порівняння коректності SQL між спеціалізованим інтерфейсом користувача та підходами самих користувачів

На основі зібраних даних було проведено порівняльний аналіз ефективності інструменту проти ручного моделювання ER-діаграм та написання SQL-коду. Результати показали, що середній час створення діаграми з використанням інструменту скоротився у 2,9 рази, а кількість синтаксичних помилок у коді зменшилася на 30%. Валідність схем в середньому на 1,9 балів вища, ніж при використанні традиційних методів. Користувачі також відзначили, що автоматична генерація SQL дозволила їм зосередитися на логіці структури даних, а не на технічних деталях.

Однак деякі учасники експерименту вказали на необхідність покращення підказок у інтерфейсі для складних зв’язків між сутностями. Ці висновки лягли в основу рекомендацій для подальшого вдосконалення системи, зокрема – інтеграції контекстно-залежних підказок та розширення бібліотеки шаблонів для типових сценаріїв використання.

ВИСНОВКИ

Робота над проектуванням та створенням бази даних вимагає багато зусиль і часу, включаючи побудову структур для подальшого переведення в код SQL. Одна з задач цієї роботи – визначити, якою мірою середовище розроблене для проектування баз даних за допомогою ШІ може допомогти експерту в досягненні цілей. Дослідження приділяє велику увагу важливості використання корисних інструментів баз даних.

У ході виконання кваліфікаційної роботи були досягнені поставлені цілі: розроблено візуальний генератор пошукових запитів NL-to-SQL, що підвищує продуктивність розробників, зменшує ймовірність помилок та сприяє стандартизації процесу проектування баз даних. Запропонована система інтегрує сучасні методи штучного інтелекту для автоматичного перетворення текстових описів у SQL-запити, що значно спрощує та прискорює створення ER-діаграм і відповідного SQL-коду. Інструмент забезпечує інтуїтивно зрозумілий графічний інтерфейс, який дозволяє користувачам без глибоких технічних знань ефективно моделювати структуру бази даних, встановлювати зв’язки між сутностями, а також підтримує різні типи ключів і унікальних груп, що підвищує точність і коректність моделі.

Проведене тестування показало, що вплив такого інтерфейсу вкрай позитивно, надаючи ряд переваг, таких як швидкість, зручність, безпека розробки, правильність написання чистого коду та ефективне повторне використання. Всі описані якості не притаманні багатьом іншим інтерфейсам, навіть беручи до уваги використання chatGPT [14].

Розроблена система може стати основою для подальших досліджень і вдосконалень у сфері автоматизації проектування баз даних із застосуванням штучного інтелекту.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Chen P. P.-S. The entity-relationship model–toward a unified view of data. *ACM Transactions on Database Systems*. 1976. Т. 1, № 1. С. 9–36. URL: https://doi.org/10.1145/320434.320440 (дата звернення: 19.05.2025).

2. Storey V. C. A selective survey of the use of artificial intelligence for database design systems. *Data & Knowledge Engineering*. 1993. Т. 11, № 1. С. 61–102. URL: https://doi.org/10.1016/0169-023x(93)90045-q (дата звернення: 19.05.2025).

3. AI Model to Generate SQL Queries from Natural Language Instructions through Voice / A. Sawant та ін. *Journal of Physics: Conference Series*. 2022. Т. 2273, № 1. С. 012014. URL: https://doi.org/10.1088/1742-6596/2273/1/012014 (дата звернення: 20.05.2025).

4. - M. S. S. MLOps: Revolutionizing AI Development and Deployment. *International Journal For Multidisciplinary Research*. 2024. Т. 6, № 5. URL: https://doi.org/10.36948/ijfmr.2024.v06i05.28794 (дата звернення: 21.05.2025).

5. Cong G., Yang J., Zhao Y. Machine Learning for Databases: Foundations, Paradigms, and Open problems. *SIGMOD/PODS '24: International Conference on Management of Data*, м. Santiago AA Chile. New York, NY, USA, 2024. URL: https://doi.org/10.1145/3626246.3654686 (дата звернення: 22.05.2025).

6. Yonghui Wu. Normalization Design of XML Database Schema for Eliminating Redundant Schemas and Satisfying Lossless Join. *IEEE/WIC/ACM International Conference on Web Intelligence (WI'04)*, м. Beijing, China. URL: https://doi.org/10.1109/wi.2004.10075 (дата звернення: 22.05.2025).

7. Shadbolt N., Berners-Lee T., Hall W. The Semantic Web Revisited. *IEEE Intelligent Systems*. 2006. Т. 21, № 3. С. 96–101. URL: https://doi.org/10.1109/mis.2006.62 (дата звернення: 23.05.2025).

8. Chaudhuri S., Dageville B., Lohman G. Self-Managing Technology in Database Management Systems. *Proceedings 2004 VLDB Conference*. 2004. С. 1243. URL: https://doi.org/10.1016/b978-012088469-8.50116-9 (дата звернення: 23.05.2025).

9. Sendas N., Rajale D. Performance Efficiency in MLOps. *The Definitive Guide to Machine Learning Operations in AWS*. Berkeley, CA, 2024. С. 223–267. URL: https://doi.org/10.1007/979-8-8688-1076-3_6 (дата звернення: 25.05.2025).

10. Wang L., Li Y., Hu J. Digital Interaction Design in the Context of Natural User Interface Based on Computer Science. *ICIEI 2022: 2022 The 7th International Conference on Information and Education Innovations*, м. Belgrade Serbia. New York, NY, USA, 2022. URL: https://doi.org/10.1145/3535735.3537770 (дата звернення: 25.05.2025).

11. NeurDB: an AI-powered autonomous data system / B. C. Ooi та ін. *Science China Information Sciences*. 2024. Т. 67, № 10. URL: https://doi.org/10.1007/s11432-024-4125-9 (дата звернення: 26.05.2025).

12. Chatziemmanouil T., Katsanos C. Accessibility Academy: Interactive Learning of the WCAG 2.1 Web Accessibility Guidelines. *2024 IEEE Global Engineering Education Conference (EDUCON)*, м. Kos Island, Greece, 8–11 трав. 2024 р. 2024. URL: https://doi.org/10.1109/educon60312.2024.10578915 (дата звернення: 26.05.2025).

13. Sayed A., Patel M. O. S. The Influence of JavaScript, on Website Development and User Experience. *International Journal for Research in Applied Science and Engineering Technology*. 2023. Т. 11, № 11. С. 87–88. URL: https://doi.org/10.22214/ijraset.2023.56284 (дата звернення: 28.05.2025).

14. Кашпур І. В., Гриньова О.Є. Візуальний генератор схем РБД засобами ШІ. *29-й Міжнародний молодіжний форум «Радіоелектроніка та молодь у XXI столітті».* Зб. матеріалів форуму. Т.6., Харків: ХНУРЕ. 2025. С. 31–33.