Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження ефективності застосування ORM та SQL підходів для доступу до баз даних у Go-додатках
(тема)

Виконав:
здобувач 2 року навчання
групи ІПЗм-23-2

Андрій ЯГНЮКОВ
(Власне ім'я, ПРІЗВИЩЕ)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-наукова

Керівник доц. Віктор КАУК
(посада, Власне ім'я, ПРІЗВИЩЕ)

Допускається до захисту
Зав. кафедри _____ Кирило СМЕЛЯКОВ
(підпис) (Власне ім'я, ПРІЗВИЩЕ)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти другий (магістерський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми освітньо-наукова програма
Освітня програма Інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Ягнюкову Андрію Юрійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи «Дослідження ефективності застосування ORM та SQL підходів для доступу до баз даних у Go-додатках»
затверджена наказом університету від 15.04.2025р. № 290 Ст
2. Термін подання студентом роботи до екзаменаційної комісії 18.06.2025р.
3. Вихідні дані до роботи опис досліджуваних ORM та SQL бібліотек, вимоги до проєктування ПЗ та БД для проведення досліджень, мова програмування Golang, СУБД PostgreSQL, середовище розробки Visual Studio Code
4. Перелік питань, що потрібно опрацювати в роботі аналіз предметної галузі, огляд літературних і наукових джерел, постановка задачі, теоретичне дослідження, практичне дослідження, висновки

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання	16.04.2025	*виконано*
2	Аналіз предметної галузі і постановка задачі	19.04.2025	*виконано*
3	Огляд й аналіз літературних, наукових джерел	24.04.2025	*виконано*
4	Вирішення багатокритеріальної задачі	28.04.2025	*виконано*
5	Проєктування ПЗ та БД	02.05.2025	*виконано*
6	Розробка ПЗ для проведення дослідження	08.05.2025	*виконано*
7	Проведення експериментального дослідження	10.05.2025	*виконано*
8	Підготовка до апробації результатів дослідження	16.05.2025	*виконано*
9	Підготовка пояснювальної записки	24.05.2025	*виконано*
10	Підготовка презентації та доповіді	28.05.2025	*виконано*
11	Перевірка на плагіат	10.06.2025	*виконано*
12	Нормоконтроль	10.06.2025	*виконано*
13	Рецензування	13.06.2025	*виконано*
14	Попередній захист	14.06.2025	*виконано*
15	Занесення диплому в електронний архів	17.06.2025	*виконано*
16	Допуск до захисту у зав. кафедри	17.06.2025	*виконано*

Дата видачі завдання 16.04.2025р.

Студент _____ Андрій ЯГНЮКОВ
(підпис)

Керівник роботи _____ доц. Віктор КАУК
(підпис) (посада, Власне ім’я, ПРІЗВИЩЕ)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка містить: 65 с., 3 рис., 10 табл., 16 джерел.

БАЗА ДАНИХ, ENT, GO, GOLANG, GORM, ORM, SQL, SQLBOILER, XORM.

Об’єктом дослідження є підходи до доступу до баз даних у Go-додатках.

Метою роботи є проведення дослідження найбільш ефективного підходу для доступу до баз даних у додатках, розроблених мовою програмування Golang.

Методами розробки та проєктування є аналіз предметної галузі, аналіз наукової інформації та літературних джерел, вибір ORM для проведення дослідження шляхом вирішення багатокритеріальної задачі, написання програми та проведення експериментальних досліджень, аналіз результатів досліджень та формування рекомендацій щодо використання того чи іншого підходу.

У результаті кваліфікаційної роботи визначено найбільш ефективний підхід до доступу до баз даних у Go-додатках, а також надано рекомендації щодо вибору та використання ORM-бібліотек і SQL-підходу.

DATABASE, ENT, GO, GOLANG, GORM, ORM, SQL, SQLBOILER, XORM.

The object of the research is the approaches to database access in Go applications.

The purpose of the work is to conduct research on the most effective approach for accessing databases in applications developed using the Go programming language.

The development and design methods are the analysis of the subject area, the analysis of scientific information and literature sources, the selection of an ORM for the study by solving a multicriteria decision-making problem, writing the program and conducting experimental research, the analysis of research results, and the formation of recommendations regarding the use of a particular approach.

As a result of the qualification work, the most effective approach to database access in Go applications has been determined, and recommendations have been provided on the selection and use of ORM libraries and the SQL-based approach.

Завідувачу кафедри
ПІ_____
(скорочена назва кафедри)
проф. Кирилу СМЕЛЯКОВУ_
(вчене звання, сласне ім'я, прізвище)

ЗАЯВА
щодо самостійності виконання кваліфікаційної роботи та можливості її публікації (та/або публікації анотації кваліфікаційної роботи) в електронному архіві відкритого доступу ElArKhNURE

Я, Ягнюков Андрій Юрійович, здобувач вищої освіти на другому (магістерському) рівні вищої освіти академічної групи ІПЗм-23-2, кафедра програмної інженерії, заявляю: моя кваліфікаційна робота на тему «Дослідження ефективності застосування ORM та SQL підходів для доступу до баз даних у Go-додатках», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в репозиторії «ElArKhNURE». Погоджуюся з авторським договором, відповідно до Положення про репозиторій ХНУРЕ «ElArKhNURE». Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений з вимогами академічної доброчесності, згідно з якими виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

18.06.2025

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

SQL – Structured Query Language

ORM – Object-Relational Mapping

API – Application Programming Interface

СУБД – Система управління базами даних

DSL – Domain-Specific Language

ОЗП – Оперативний запам'ятовуючий пристрій

GC – Garbage Collection

ВСТУП

На теперішній час мова програмування Go активно використовується для розробки високонавантажених систем та мікросервісів. Зростаюча популярність Go зумовлює потребу в оптимальному виборі підходу до роботи з базами даних, проте відсутність комплексних досліджень ефективності ORM-бібліотек порівняно з SQL-підходом ускладнює прийняття обґрунтованих архітектурних рішень.

Метою роботи є визначення найбільш ефективного підходу для доступу до баз даних у додатках, розроблених мовою програмування Go.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

- провести огляд й аналіз існуючих підходів до взаємодії з базами даних;
- визначити систему критеріїв оцінювання та ключові метрики для проведення порівняльного аналізу;
- спроєктувати та реалізувати програмний додаток для проведення експериментальних досліджень;
- виконати вимірювання продуктивності кожного підходу за визначеними метриками;
- сформулювати рекомендації щодо доцільності застосування різних підходів.

Об’єктом дослідження є підходи до взаємодії з базами даних у додатках, розроблених мовою програмування Go.

Предметом дослідження є методи взаємодії з базами даних, зокрема ORM-бібліотеки та SQL-підхід.

Для досягнення поставленої мети застосовуються такі методи дослідження:

- аналіз предметної галузі та літературно-наукової інформації: вивчення існуючих підходів до взаємодії з базами даних та їх особливостей;
- багатокритеріальний аналіз: обґрунтування вибору ORM-бібліотек для проведення дослідження на основі визначених критеріїв;

- експериментальні дослідження: розробка програмного додатка та проведення експериментів для порівняння продуктивності різних підходів на основі визначених метрик;
- систематизація даних: формування рекомендацій щодо доцільності використання того чи іншого підходу.

У результаті було виконано порівняльний аналіз ефективності ORM-підходу та прямого використання SQL-запитів у додатках на базі Go. Отримані результати дозволили визначити оптимальний інструмент для забезпечення високої продуктивності та масштабованості систем, що, своєю чергою, сприятиме підвищенню якості програмного забезпечення.

Результати кваліфікаційної роботи було представлено на молодіжній конференції MIT@AISm-2025, яка проводилася в межах 1-ї Міжнародної науково-практичної конференції «СУЧАСНІ ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ТА СИСТЕМИ ШТУЧНОГО ІНТЕЛЕКТУ MIT@AIS-2025».

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз існуючих підходів

Сучасна розробка програмного забезпечення для взаємодії з реляційними базами даних у Go базується на двох основних підходах. Перший підхід – це використання стандартного пакета database/sql для написання SQL-запитів, а другий – застосування ORM-бібліотек, таких як GORM, XORM, Ent та SQLBoiler. Кожен із цих підходів має свої переваги та недоліки, що впливають на вибір оптимального рішення.

Пакет database/sql є вбудованим засобом мови Golang для взаємодії з СУБД. Використання цього підходу забезпечує повний контроль над формуванням SQL-запитів, що дозволяє оптимізувати кожен аспект виконання запиту.

Основними перевагами такого підходу є:

- висока продуктивність: оскільки запити формуються вручну, то можна детально оптимізувати їх під конкретну СУБД, мінімізуючи накладні витрати;
- гнучкість: дозволяє використовувати специфічні для СУБД функції, оптимізувати запити для великих обсягів даних, реалізовувати складні алгоритми обробки та агрегації;
- універсальність: знання стандартного SQL є переносимим, тому набуті навички застосовуються не лише в Go, але й в інших мовах програмування та середовищах.

Проте робота з чистими SQL-запитами має і недоліки. Основним є необхідність написання великої кількості шаблонного коду для обробки результатів запитів, управління транзакціями та обробки помилок. Крім того, через відсутність абстракції ризик виникнення помилок набагато вищий. Також, у великих проєктах, де структура бази даних часто змінюється, підтримка коду може стати складною, оскільки кожна зміна в схемі вимагає відповідного коригування SQL-запитів [1].

ORM-бібліотеки створені з метою підвищення продуктивності розробки шляхом надання можливості працювати з об’єктами, а не з чистими SQL-запитами. В Golang найбільш популярними ORM-бібліотеками є GORM, XORM, Ent та SQLBoiler. Ці інструменти забезпечують додатковий рівень абстракції, що спрощує створення, оновлення та читання даних, а також автоматизує ряд процесів, таких як міграції бази даних та обробка транзакцій.

GORM – це найпопулярніша ORM-бібліотека в Go. Вона побудована поверх стандартного database/sql і використовує підхід «code-first», де моделі визначаються за допомогою структур Go з відповідними тегами.

Серед основних переваг GORM можна виділити:

- зручність і швидкість розробки: більшість операцій виконується за допомогою викликів методів, що зменшує написання повторюваного коду;
- автоматичне створення та міграція схем: дозволяє швидко синхронізувати моделі Go з таблицями в базі даних;
- гнучкість у виконанні чистих SQL: при необхідності можна звернутися до виконання запитів напряму, зберігаючи контроль над продуктивністю.

Серед недоліків є додатковий рівень абстракції, який іноді може призводити до зниження швидкодії, особливо при роботі з великими об’ємами даних. Також, специфіка роботи з асоціаціями та зв’язками між таблицями може вимагати детального вивчення документації, що створює певну криву навчання [2].

XORM є альтернативою GORM і орієнтований на простоту використання та високу продуктивність. XORM використовує теги структур для визначення моделей даних, але його API має більш лаконічний синтаксис, що дозволяє швидко розпочати розробку.

Основні переваги XORM:

- оптимізація операцій запису: у незалежних тестах XORM часто демонструє кращу продуктивність при вставці та оновленні даних;

– простота конфігурації: завдяки мінімальному набору налаштувань, впровадження XORM у проєкт здійснюється швидко.

До недоліків можна віднести менший набір розширених можливостей порівняно з GORM, а також менш активну спільноту, що іноді уповільнює впровадження нових можливостей [3].

Ent – це сучасна ORM-бібліотека, що відрізняється використанням генерації коду на етапі компіляції. Завдяки цьому Ent забезпечує статичну типізацію моделей, що дозволяє виявляти помилки ще на етапі компіляції.

Основні переваги Ent:

– типобезпечність: запити та операції формуються з використанням типізації, що знижує ймовірність помилок під час виконання;
– потужний механізм роботи зі зв’язками: завдяки спеціальному DSL, дозволяє описувати складні взаємозв’язки між сутностями;
– оптимальна продуктивність: згенерований код майже не використовує рефлексію, що дозволяє знизити накладні витрати під час виконання запитів.

Серед недоліків варто зазначити високий поріг входження. Доводиться засвоювати специфіку генерації коду та концепції, пов’язані з описом схем, що може потребувати додаткового часу на навчання. Крім того, генерація коду збільшує час компіляції проєкту, що може стати критичним у великих системах [4].

SQLBoiler реалізує принцип «database-first», тобто спочатку створюють схему бази даних, а потім генерується відповідний ORM-код на основі цієї схеми.

Основні переваги SQLBoiler:

– повна відповідність коду схемі: згенеровані моделі точно відображають структуру таблиць, що забезпечує високу типобезпечність та зменшує ризик синтаксичних помилок;
– висока продуктивність: завдяки генерації коду, виконання запитів здійснюється з мінімальними накладними витратами.

Основним недоліком є необхідність повторної генерації коду при кожній зміні схеми бази даних. У проєктах, де база даних постійно змінюється, це додає додатковий етап у процес розробки. Крім того, налаштування генератора може бути складнішим, що вимагає детального ознайомлення з документацією [5].

Аналізуючи вищенаведені підходи, можна відзначити, якщо критично важлива максимальна продуктивність і потрібен контроль над SQL-запитами, доцільніше використовувати чистий SQL. Проте у випадках, коли пріоритетом є швидка розробка, зручність підтримки коду і можливість швидкого масштабування функціональності, ORM-бібліотеки є більш кращим рішеннями.

1.2 Виявлення проблем та актуалізація рішень

Попри переваги кожного з підходів, їх застосування супроводжується рядом проблем, які стають актуальними при розробці сучасних Go-додатків. Ці проблеми можна згрупувати за кількома ключовими напрямками: продуктивність, гнучкість та підтримка, сумісність із СУБД, а також складність навчання та використання.

Однією з найважливіших проблем є зниження швидкодії додатка при використанні ORM-бібліотек через додатковий рівень абстракції. Виконання складних операцій, таких як масові вставки або оновлення, через ORM може бути значно повільнішим у порівнянні з ручним написанням SQL-запитів. Ця проблема стає особливо актуальною у високонавантажених системах.

Хоча ORM значно спрощують роботу зі стандартними операціями, вони часто не дозволяють ефективно реалізувати складні запити, що вимагають використання специфічних можливостей СУБД. Обмеження API можуть змушувати застосовувати обхідні шляхи або писати чистий SQL для реалізації нетривіальних операцій.

Для розробників, які вперше стикаються з ORM, може бути досить складно опанувати специфіку роботи з конкретною бібліотекою. Кожен з розглянутих ORM має свої особливості конфігурації, визначення моделей і побудови запитів. Для великих проєктів, де від зміни архітектури залежить швидкість розробки, висока

крива навчання може стати значною перешкодою. Навпаки, використання чистих SQL-запитів вимагає глибоких знань SQL, але забезпечує універсальність і переносимість цих знань між різними проєктами.

Отже, актуальність дослідження ефективності застосування ORM та SQL підходів у Go-додатках полягає в необхідності знаходження компромісу між двома різними підходами. В умовах стрімкого розвитку технологій та зростаючих вимог до продуктивності сучасних систем важливо оцінити, у яких випадках краще використовувати повнофункціональні ORM, а коли – повністю покладатися на ручне написання SQL-запитів.

Враховуючи це, подальше дослідження має зосередитися на порівняльному аналізі ефективності зазначених ORM-бібліотек та прямого використання SQL. Такий аналіз дозволить не лише визначити оптимальний інструмент, але й сформувати рекомендації для розробників, які прагнуть поєднати зручність високорівневих бібліотек із максимальною продуктивністю системи.

2 ОГЛЯД Й АНАЛІЗ ЛІТЕРАТУРНИХ, НАУКОВИХ ДЖЕРЕЛ

2.1 Критерії вибору джерел

Для проведення дослідження було відібрано наукові роботи, які повністю висвітлюють аспекти використання ORM і SQL підходів у роботі з базами даних. Відбір джерел здійснювався відповідно до чітко визначених критеріїв, що забезпечують наукову обґрунтованість і актуальність отриманих даних.

Основними критеріями вибору наукових джерел є:

- авторитетність: використання публікацій з високою репутацією серед наукової спільноти, таких як рецензовані журнали та матеріали міжнародних конференцій, що забезпечує довіру до результатів досліджень і гарантує їх наукову цінність;
- актуальність: відбір джерел, які відображають сучасний стан розвитку технологій та враховують останні досягнення в галузі, створених переважно протягом останніх років, що дозволяє врахувати новітні тенденції в розробці програмного забезпечення;
- об'єктивність: використання джерел, що надають збалансовану інформацію на основі фактичних даних і детального аналізу, без суб'єктивних оцінок, забезпечуючи прозорість висновків;
- достовірність: відбір робіт, дані яких підкріплені реальними експериментами, визнаними методологіями або ґрунтовними літературними оглядами, що сприяє формуванню обґрунтованих висновків і можливості застосування отриманих знань у практичній діяльності.

2.2 Аналіз літератури

Аналіз літературних джерел є важливим етапом дослідження, який дозволяє оцінити переваги та недоліки ORM та SQL у контексті роботи з базами даних. У дослідженні [6] наголошується на високій продуктивності SQL-підходу для складних

операцій, що пояснюється відсутністю додаткового рівня абстракції. Проте ORM демонструють значну перевагу в зручності розробки, зменшуючи кількість повторюваного коду, що було підтверджено результатами експериментів, представленими у дослідженні [8].

Дослідження [6] і [9] включали емпіричний аналіз виконання типових операцій на реальних наборах даних.

У результаті експериментів було встановлено, що:

– чистий SQL значно перевершує ORM за швидкістю обробки складних запитів, особливо при роботі з великими обсягами даних;
– ORM забезпечує оптимальну продуктивність для виконання стандартних операцій, таких як створення, оновлення та читання даних, що робить цей підхід найкращим для швидкого прототипування.

У дослідженні [7] основну увагу було приділено вимірюванню таких показників, як час виконання запитів та обсяг споживаної пам'яті.

Застосовуючи великі набори даних, автори дійшли висновку, що:

– SQL-підхід є оптимальним вибором для виконання складних обчислень, забезпечуючи високу продуктивність;
– ORM забезпечує простоту впровадження, яка є критично важливою для малих і середніх проєктів, де швидкість розробки має пріоритет.

У дослідженні [8] проведено аналіз проблем надлишкового доступу до даних, характерних для ORM. Дослідження показало, що неефективна організація запитів може значно знижувати продуктивність. Застосовуючи оптимізацію, таку як стратегічна вибірка даних або уникнення проблеми N+1, було досягнуто суттєвого скорочення часу виконання запитів. Це підкреслює важливість правильної конфігурації ORM, яка здатна мінімізувати накладні витрати. Водночас чистий SQL, забезпечуючи прямий контроль над запитами, уникає подібних недоліків.

На основі розглянутих джерел можна зробити висновок, що вибір між ORM та SQL залежить від конкретних потреб. Для невеликих систем із типовими операціями

ORM є більш зручним вибором, тоді як для високонавантажених додатків SQL-підхід забезпечує вищу продуктивність. Важливим аспектом є можливість оптимізації ORM, яка дозволяє адаптувати його для складніших сценаріїв.

Таким чином, SQL виявляє себе як більш ефективний підхід для виконання складних операцій, тоді як ORM пропонує суттєві переваги в зручності розробки та зменшенні витрат часу на реалізацію стандартних операцій. Подальше дослідження цих підходів може бути спрямоване на аналіз їх ефективності у контексті реального Go-додатка із врахуванням ресурсомісткості та продуктивності.

2.3 Оцінка актуальності та новизни

Дослідження [6, 7, 8, 9] є актуальними у контексті зростаючих вимог до продуктивності та оптимізації баз даних, які висуваються сучасними програмними системами. Актуальність цих досліджень зумовлена такими факторами:

- необхідність вибору ефективних підходів до роботи з даними, які здатні відповідати потребам масштабованих додатків;
- фокус на практичному застосуванні ORM і SQL, що підкреслює їхню важливість.

Емпіричні дослідження, такі як [6] і [9], пропонують новаторські підходи до оцінки продуктивності та ресурсомісткості в умовах реальних сценаріїв. Ці дослідження демонструють, що:

- використання SQL дозволяє досягати високої ефективності при виконанні складних запитів;
- ORM забезпечують простоту впровадження й підтримки додатків завдяки автоматизації роботи з даними.

Важливим внеском робіт є деталізовані результати, які сприяють обґрунтованому вибору методів для конкретних умов використання.

Новизна досліджень полягає у застосуванні різних методологій, таких як аналіз надлишкового доступу до даних у дослідженні [8], що дає змогу краще розуміти

недоліки ORM і способи їхньої оптимізації. Подібний підхід відкриває нові можливості для адаптації підходів залежно від конкретних потреб проєктів.

Ці дослідження створюють основу для подальших експериментів із застосуванням сучасних інструментів та технологій, особливо в контексті мови програмування Golang. Їхнє значення полягає у здатності адаптувати результати до реальних умов, що дозволяє приймати рішення на основі емпіричних даних і точних аналізів.

Таким чином, дослідження [6, 7, 8, 9] забезпечують цінний внесок у розвиток сучасних підходів до роботи з базами даних, інтегруючи новаторські методології та пропонуючи практичні рекомендації для вдосконалення процесу розробки програмного забезпечення.

2.4 Висновки з огляду

На основі проведеного аналізу літературно-наукових джерел можна зробити декілька ключових висновків, які стосуються застосування ORM та SQL підходів у сучасних програмних систем.

По-перше, чистий SQL демонструє значну перевагу у складних сценаріях, які включають виконання великих аналітичних запитів чи обробку обширних обсягів даних. Основною причиною цього є відсутність додаткового рівня абстракції, що дозволяє SQL напряму працювати з базою даних, забезпечуючи оптимальну швидкість та ефективність виконання запитів.

По-друге, ORM є ефективним інструментом для прискорення розробки програмного забезпечення, оскільки забезпечує автоматизацію генерації запитів і спрощує управління базою даних. Це робить ORM оптимальним вибором, де пріоритетним є зменшення часу розробки і забезпечення легкості впровадження.

Водночас ORM має певні обмеження, зокрема:

– проблеми з продуктивністю та масштабованістю, які стають критичними у випадку роботи з великими базами даних;

- недоліки при виконанні складних запитів, що вимагають особливої оптимізації;
- проблема надлишкового доступу до даних, яка може значно впливати на продуктивність.

Для подальшого дослідження доцільним є створення експериментального Go-додатка, який реалізує підходи як з використанням ORM, так і SQL. Це дозволить провести порівняння ефективності за різними метриками, які повинні зосереджуватися на аналізі типових сценаріїв і запитів, які найчастіше зустрічаються у реальних додатках. Такий підхід допоможе робити обґрунтовані вибори і досягати оптимального балансу між продуктивністю та зручністю роботи з базами даних у Go-додатках.

3 ПОСТАНОВКА ЗАДАЧІ

3.1 Формулювання задачі

Аналіз предметної галузі та огляд літературно-наукових джерел показали, що існують два основні способи взаємодії з СУБД: через ORM-бібліотеки або шляхом виконання SQL-запитів, які різняться рівнем абстракції, швидкістю розробки та ступенем контролю над виконанням запитів. Хоча відомо, що ORM-інструменти забезпечують вищий рівень абстракції і підвищують продуктивність, але часто поступаються швидкістю виконання операцій прямим SQL-запитам, особливо при зростанні обсягу даних, залишається недостатньо з'ясованим, як саме використання ORM впливає на продуктивність Go-додатків порівняно з прямим SQL-підходом. У контексті мови Go бракує досліджень з цього питання, що створює невизначеність при виборі підходу доступу до даних, тому постає проблема необхідності дослідження та порівняння ефективності ORM-підходу і традиційного SQL-підходу для роботи з базами даних.

Метою роботи є визначення найбільш ефективного підходу для доступу до баз даних у додатках, розроблених мовою програмування Go. Для досягнення цієї мети передбачено розв’язати такі завдання:

- на основі багатокритеріального аналізу обрати ORM-бібліотеки для детального дослідження ефективності;
- розробити методику експериментального дослідження, зокрема визначити ключові метрики та реалізувати Go-додаток з однаковою функціональністю для ORM та SQL підходів;
- провести серію експериментів, вимірявши обрані метрики для кожного підходу;
- проаналізувати отримані результати та сформулювати висновки та рекомендації щодо доцільності використання ORM-бібліотек та SQL-підходу в Go-додатках.

3.2 Обґрунтування вибору методів дослідження

Для виконання поставлених завдань використано поєднання теоретичного та експериментального підходів. Запропонована методологія дослідження складається з кількох основних етапів.

На першому етапі здійснено багатокритеріальний аналіз доступних ORM-бібліотек. Такий аналіз дозволяє систематично оцінити альтернативи за низкою критеріїв та звузити вибір до найперспективніших. Обґрунтований попередній відбір забезпечує репрезентативність подальших експериментів.

Наступним етапом стало експериментальне оцінювання продуктивності обраних підходів. Було проведено серію контрольованих експериментів, у межах яких ORM-підхід і прямі SQL-запити виконували однакові операції над базою даних. Такий підхід дозволив отримати кількісні показники продуктивності та порівняти ефективність методів в ідентичних умовах.

3.3 Обмеження дослідження

Під час дослідження встановлено такі обмеження для його проведення:

- область застосування: робота охоплює лише реляційні СУБД. Інші типи баз даних та альтернативні способи доступу до даних не розглядаються;
- середовище та СУБД: усі експерименти здійснюються на одній обраній системі керування базами даних. Отримані результати характерні саме для цієї СУБД та конфігурації;
- інфраструктура: вимірювання проводяться в однакових технічних умовах. Абсолютні значення метрик можуть відрізнятися за інших умов, тому акцент робиться на порівнянні показників між підходами.

3.4 Необхідні ресурси

Для реалізації поставлених завдань і проведення експериментів потрібні такі ресурси:

- програмні засоби: мова програмування Go, СУБД PostgreSQL, ORM-бібліотеки, стандартний пакет SQL, Docker для контейнеризації та інструмент golang-migrate для міграцій;
- апаратне забезпечення: комп’ютер з процесором Intel Core i5-9400F (6 ядер, 6 потоків) та 24 ГБ ОЗП;
- тестові дані: набір даних достатнього обсягу для заповнення таблиць, що розгортається через SQL-скрипти;
- інформаційні матеріали: офіційна документація, релевантні наукові статті про продуктивність ORM і прямого SQL;
- засоби моніторингу та профілювання: вбудовані Go-пакети time та runtime.

4 ТЕОРЕТИЧНЕ ДОСЛІДЖЕННЯ

4.1 Змістовно сформульована задача багатокритеріального вибору

Перед початком експериментального дослідження необхідно здійснити багатокритеріальний відбір серед зазначених ORM-бібліотек, щоб визначити ті, що мають найбільший потенціал для подальшого аналізу їх ефективності при взаємодії з базами даних. Нижче наведені критерії відбору [10]:

- документація та навчальні матеріали;
- кількість підтримуваних СУБД;
- популярність серед розробників на GitHub;
- легкість інтеграції та використання;
- підтримка спільноти та оновлення.

Вибір найефективніших ORM-бібліотек за зазначеними критеріями дозволить забезпечити об'єктивність і достовірність результатів аналізу ефективності їх застосування у Go-додатках порівняно з прямим SQL-підходом.

4.2 Опис множини альтернатив для задачі вибору

Нижче наведено опис усіх альтернатив, що розглядаються у межах задачі вибору.

Бібліотека GORM: підходить для проєктів середнього масштабу, де важлива швидка розробка та підтримка складних транзакцій і асоціацій. Вона забезпечує гнучкість у роботі з базами даних, дозволяючи зосередитись на бізнес-логіці, а не на написанні великої кількості рутинного SQL-коду.

Переваги:

- широкий функціонал, включаючи підтримку асоціацій, транзакцій та автоматизованих міграцій;
- потужний механізм розширення, який дозволяє адаптувати бібліотеку під специфічні потреби проєкту.

Недоліки:

- додатковий рівень абстракції може негативно впливати на продуктивність, особливо при роботі з великими обсягами даних;
- робота зі складними запитами іноді вимагає глибшого вивчення документації.

Бібліотека XORM: оптимальний вибір для невеликих проєктів або стартапів, де головним завданням є швидка інтеграція та простота використання. Вона дозволяє швидко розпочати роботу завдяки лаконічному API і мінімальній конфігурації, але може бути не найкращою для великих або надскладних систем.

Переваги:

- простий і інтуїтивний API, який дозволяє оперативно розпочати розробку;
- мінімальна конфігурація, що забезпечує швидке впровадження в проєкт.

Недоліки:

- обмежений набір функціональних можливостей порівняно з іншими ORM-бібліотеками;
- менша активність спільноти може впливати на оперативність отримання нових функцій та підтримки.

Бібліотека Ent: підходить для складних і масштабованих систем, де критично важлива типобезпечність та робота із складними взаємозв’язками між сутностями. Вона орієнтована на команди досвідчених розробників, готових вкласти час у вивчення специфіки генерації коду.

Переваги:

- генерація типобезпечного коду, що знижує ризик помилок під час виконання операцій;
- потужний механізм роботи зі зв’язками між сутностями, який дозволяє описувати складні взаємозв’язки.

Недоліки:

– високий поріг входу: потребує детального вивчення специфіки генерації коду та опису схем;
– збільшення часу компіляції може стати критичним у великих системах.

Бібліотека SQLBoiler: доцільно використовувати в проєктах, де важлива точна відповідність між структурою бази даних і програмним кодом. Це особливо корисно для систем із чіткими вимогами до схеми, де продуктивність і типобезпечність мають першорядне значення.

Переваги:

– генерація ORM-коду, яка точно відображає структуру бази даних, забезпечує високу типобезпечність;
– висока відповідність між програмним кодом і базою даних сприяє зниженню ризику синтаксичних помилок.

Недоліки:

– необхідність регулярної перегенерації коду при зміні схеми бази даних;
– високий поріг входу для початківців через специфіку генерації коду та налаштувань.

4.3 Опис множини критеріїв для задачі багатокритеріального вибору

Документація та навчальні матеріали: критерій оцінює якість офіційної документації та наявність додаткових навчальних ресурсів, таких як приклади коду, посібники та інструкції. Це забезпечує розробникам зручність і швидкість освоєння інструмента.

Кількість підтримуваних СУБД: критерій визначає, наскільки бібліотека здатна інтегруватися з різними типами СУБД. Це важливо для гнучкості в роботі та розширення сфер застосування інструмента.

Популярність серед розробників на GitHub: цей критерій оцінює загальну популярність бібліотеки в спільноті розробників, що вказує на її довіру, поширеність і підтримку.

Легкість інтеграції та використання: критерій описує зручність впровадження ORM у додаток та рівень простоти роботи з її API. Це впливає на ефективність роботи розробників та загальний час впровадження.

Підтримка спільноти та оновлення: критерій відображає рівень активності розробників та спільноти, частоту оновлень, а також доступність підтримки. Це забезпечує стабільність і актуальність бібліотеки в довготривалих проєктах.

4.4 Опис та аналіз шкал за кожним з обраних критеріїв

Розглянемо кожний критерій окремо та визначимо підходящу шкалу оцінок для кожного з них, де всі значення були сформовані з офіційних документацій [2, 3, 4, 5].

Документація та навчальні матеріали: оцінка якості та доступності документації й навчальних матеріалів для розробників, включаючи приклади, посібники та відеоуроки. Чим краща документація, тим простіше розробникам використовувати ORM-бібліотеку.

Тип шкали: шкала порядку.

Приклади значень:

- 1 бал: документація поверхова або відсутня, відсутність навчальних матеріалів;
- 2 бали: документація достатня для базового використання, але складна у випадках нестандартних сценаріїв;
- 3 бали: детальна документація, якісні навчальні матеріали, приклади коду.

Кількість підтримуваних СУБД: вимірюється у кількості баз даних, які підтримує бібліотека. Більша кількість підтримуваних СУБД розширює спектр застосування ORM.

Тип шкали: шкала відношень.

Приклади значень:

– 1-2 БД: низька підтримка;

– 3-5 БД: середня підтримка;

– більше 5 БД: висока підтримка.

Популярність серед розробників на GitHub: вимірюється у кількості тисяч зірок на сторінці бібліотеки в GitHub. Відображає рівень зацікавленості та довіри спільноти розробників.

Тип шкали: шкала відношень.

Приклади значень:

– 1-5 тисяч зірок: низька популярність;

– 6-15 тисяч зірок: середня популярність;

– більше 15 тисяч зірок: висока популярність.

Легкість інтеграції та використання: оцінка простоти налаштування, зрозумілості API та наявності прикладів використання. Чим простіше інтегрувати ORM, тим вищий рівень легкості.

Тип шкали: шкала порядку.

Приклади значень:

– 1 бал: складна інтеграція, незрозумілий API;

– 2 бали: задовільна інтеграція із певними складнощами;

– 3 бали: проста інтеграція, зрозумілий API, доступність прикладів.

Підтримка спільноти та оновлення: визначає активність спільноти, регулярність оновлень бібліотеки та швидкість вирішення проблем. Чим активніша спільнота, тим більше підтримки може отримати розробник.

Тип шкали: шкала порядку.

Приклади значень:

– 1 бал: рідкісні оновлення, низька активність спільноти;

– 2 бали: періодичні оновлення, обмежена активність;

– 3 бали: регулярні оновлення, активна спільнота.

4.5 Векторний опис та аналіз Парето-оптимальності альтернатив

Після визначення критеріїв та шкали оцінок, перейдемо до векторного опису альтернатив за обраними критеріями.

В таблиці 4.1 наведено векторний опис альтернатив за обраними критеріями.

Таблиця 4.1 – Векторний опис альтернатив за обраними критеріями (таблиця виконана самостійно)

Критерії / Альтернативи	GORM	XORM	SQLBoiler	Ent
Документація та навчальні матеріали	3	2	2	3
Кількість підтримуваних СУБД	5	7	5	6
Популярність серед розробників на GitHub	37.3	6.7	6.8	15.8
Легкість інтеграції та використання	3	2	2	3
Підтримка спільноти та оновлення	3	2	3	3

Застосуємо принцип Парето для визначення найгірших ORM-бібліотек, який наведено в таблиці 4.2.

Таблиця 4.2 – Аналіз за принципом Парето (таблиця виконана самостійно)

Критерії / Альтернативи	GORM	Ent
Документація та навчальні матеріали	3	3
Кількість підтримуваних СУБД	5	6
Популярність серед розробників на GitHub	37.3	15.8
Легкість інтеграції та використання	3	3
Підтримка спільноти та оновлення	3	3

Бібліотека GORM демонструє сильні результати за майже всіма критеріями, але поступається іншим альтернативам за кількістю підтримуваних баз даних. Завдяки збалансованим показникам за основними критеріями вона входить до множини Парето-оптимальних альтернатив.

Бібліотека XORM демонструє найвищий результат за кількістю підтримуваних баз даних, але програє за всіма іншими критеріями. Через це вона також виключається зі списку Парето-оптимальних альтернатив.

Бібліотека Ent також демонструє сильні результати за майже всіма критеріями, але поступається лише GORM у популярності серед розробників на GitHub. Незважаючи на це, вона залишається Парето-оптимальною альтернативою.

Бібліотека SQLBoiler має середні результати за всіма критеріями, поступаючись GORM і Ent. Через ці обмеження SQLBoiler виключається зі списку Парето-оптимальних альтернатив.

Отже, до множини Парето-оптимальних альтернатив входять GORM та Ent, які мають найбільш збалансовані результати за всіма критеріями.

4.6 Нормування оцінок за шкалами

Виконаємо нормування оцінок за формулою 4.1, бо критерії були приведенні до принципу оптимальності «за максимумом»:

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{4.1}$$

де x – це початкове значення показника для альтернативи,

x_{min} – це найменше значення показника серед усіх альтернатив,

x_{max} – це найбільше значення показника серед усіх альтернатив.

У таблиці 4.3 наведено нормування оцінок.

Таблиця 4.3 – Нормування оцінок (таблиця виконана самостійно)

Критерії / Альтернативи	GORM	Ent
Документація та навчальні матеріали	1	1
Кількість підтримуваних СУБД	0	1
Популярність серед розробників на GitHub	1	0

Кінець таблиці 4.3

Легкість інтеграції та використання	1	1
Підтримка спільноти та оновлення	1	1

У результаті отримано діапазон кожного критерія від 0 до 1.

4.7 Вибір згорткової моделі

В якості згорткової моделі обрано лінійну адитивну згортку з ваговими коефіцієнтами. Ця модель дозволяє враховувати різну важливість критеріїв і ефективно комбінувати їх у єдину оцінку. Важливість критеріїв визначається наступним чином:

- документація та навчальні матеріали є важливим критерієм, оскільки якісна документація сприяє швидкому освоєнню та ефективному використанню бібліотеки. Призначається 4 бали;
- кількість підтримуваних баз даних є важливим показником, адже підтримка різноманітних СУБД забезпечує гнучкість у виборі бази даних для конкретного проєкту. Призначається 5 балів;
- популярність серед розробників на GitHub визначає рівень довіри до бібліотеки. Призначається 3 бали;
- легкість інтеграції та використання впливає на зручність впровадження бібліотеки у проєкт та швидкість її адаптації. Призначається 6 балів;
- підтримка спільноти та оновлення забезпечує стабільність та актуальність бібліотеки, що важливо для довготривалих проєктів. Призначається 4 бали.

Отже, для обраної згорткової моделі і критеріїв, визначення корисності альтернатив буде виглядати наступною формулою 4.2:

$$Z^* = \max_{i=1,m} \sum_{j=1}^{n} \beta_j \alpha_{ij} \tag{4.2}$$

де β_j – це ваговий коефіцієнт,

a_{ij} – це нормоване значення для *i*-го методу та *j*-го критерію.

Виконаємо розрахунки:

– для GORM:

$$\frac{4}{22} \times 1 + \frac{5}{22} \times 0 + \frac{3}{22} \times 1 + \frac{6}{22} \times 1 + \frac{4}{22} \times 1 = 0{,}773;$$

– для Ent:

$$\frac{4}{22} \times 1 + \frac{5}{22} \times 1 + \frac{3}{22} \times 0 + \frac{6}{22} \times 1 + \frac{4}{22} \times 1 = 0{,}864.$$

У результаті виконаних обчислень було отримано оцінки альтернатив, які далі використовуються для аналізу отриманих результатів.

4.8 Аналіз отриманих результатів

Серед чотирьох розглянутих ORM-бібліотек визначено дві Парето-оптимальні альтернативи – GORM та Ent. Застосування лінійної адитивної згортки з ваговими коефіцієнтами показало, що Ent отримала найвищу інтегральну оцінку.

Ent демонструє збалансовані високі показники: відмінну документацію, підтримку значної кількості СУБД, легку інтеграцію та хорошу підтримку спільноти. Хоча за популярністю на GitHub поступається GORM, суттєва вага інших критеріїв забезпечила її перевагу.

Результати визначають Ent та GORM як найперспективніші ORM-бібліотеки для експериментального дослідження. Ці бібліотеки будуть порівняні з прямим SQL-підходом у практичній частині для комплексної оцінки ефективності різних методів доступу до баз даних у Go-додатках.

5 ПРАКТИЧНЕ ДОСЛІДЖЕННЯ

5.1 Проєктування ПЗ

У рамках експериментального дослідження реалізовано Go-додаток з трьома модулями, що моделюють предметну область електронної комерції – процес купівлі-продажу товарів і послуг через Інтернет. При цьому код побудовано за принципами об’єктно-орієнтованого програмування: усі ключові сутності представлені як окремі структури, які використовуються у всіх модулях. Розробка здійснювалася мовою програмування Go із використанням стандартних вбудованих пакетів, що забезпечить надійність та високий рівень оптимізації коду. Розділення на окремі модулі дозволило підтримувати спільну модель даних і утиліти, водночас показуючи різні підходи до доступу до бази даних [11]. Для кращої наочності була розроблена діаграма пакетів, яка зображена на рисунку 5.1.

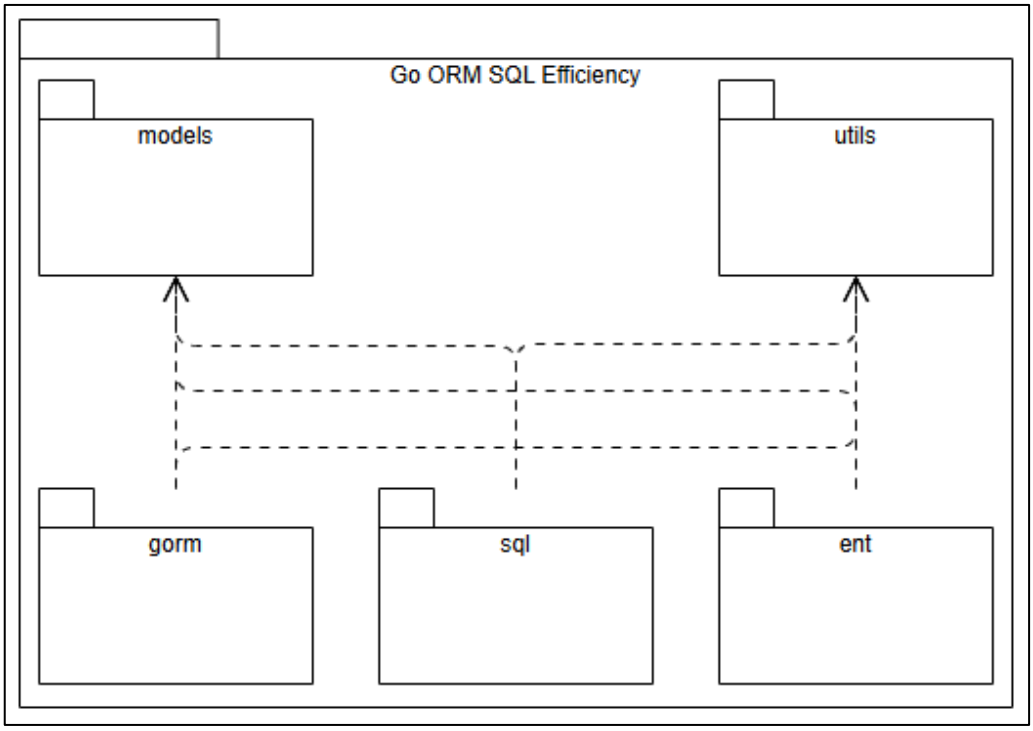

Рисунок 5.1 – Діаграма пакетів (рисунок виконаний самостійно)

Два модулі реалізовано з використанням ORM-підходів – на базі GORM та Ent, які були обрані як найперспективніші відповідно до багатокритеріального аналізу.

Третій модуль побудовано на основі стандартного пакета database/sql для прямого формування SQL-запитів. Такий підхід дає змогу порівняти ефективність і складність розробки кожного методу в єдиному кодовому середовищі.

У якості СУБД обрано PostgreSQL, яке завдяки своїй популярності здобуло перше місце в опитуванні StackOverflow 2024 серед розробників [12, 13]. Для всіх трьох модулів використано спільну базу даних у середовищі Docker, що забезпечило стандартизоване та ізольоване тестове середовище й запобігло вплив зовнішніх факторів на результати [14]. Міграції налаштовувалися через golang-migrate, що дозволило автоматизувати процес створення й оновлення схеми та гарантувало однорідність налаштувань у кожному тестовому сценарії [15].

5.2 Проєктування БД

Основна увага приділяється роботі з базою даних, тому моделювання бази даних є ключовим етапом у розробці інтернет-магазинів та інших платформ.

Перейдемо до створення ER-діаграми, де необхідно описати кожну сутність разом з її атрибутами. ER-діаграма виступає як візуальний інструмент моделювання, який демонструє структуру бази даних та взаємозв'язки між її елементами. Вона є незамінною при проєктуванні, оскільки допомагає зрозуміти організацію даних і зв'язки між ними ще до початку реалізації (див. рис. 5.2).

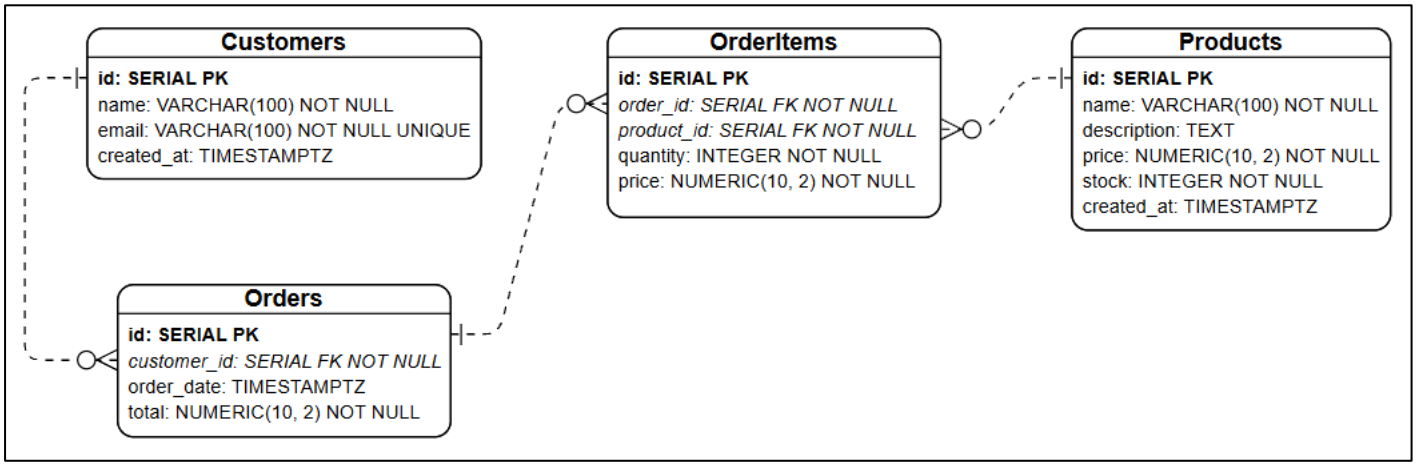

Рисунок 5.2 – ER-діаграма бази даних (рисунок виконаний самостійно)

Таблиця Клієнти (Customers): таблиця зберігає інформацію про кожного клієнта, де кожен запис має свій унікальний ідентифікатор, ім’я, електронну пошту, що повинна бути унікальною, та дату створення запису.

Таблиця Продукти (Products): у таблиці зберігається інформація про кожен товар, включаючи його унікальний ідентифікатор, назву, опис, ціну, кількість товару на складі та дату додавання запису.

Таблиця Замовлення (Orders): таблиця призначена для збереження даних про кожне замовлення, де кожне замовлення має свій унікальний ідентифікатор, дату оформлення, загальну вартість та посилання на клієнта, який зробив замовлення.

Таблиця Позиції замовлення (OrderItems): таблиця відповідає за зберігання інформації про окремі позиції в кожному замовленні, включаючи кількість замовлених товарів та їх ціну, а також посилання на відповідне замовлення і товар.

Наступним кроком є розгортання самої бази даних у контейнері Docker. Використання контейнера гарантує, що налаштування PostgreSQL завжди будуть однаковими незалежно від середовища (див. рис. 5.3).

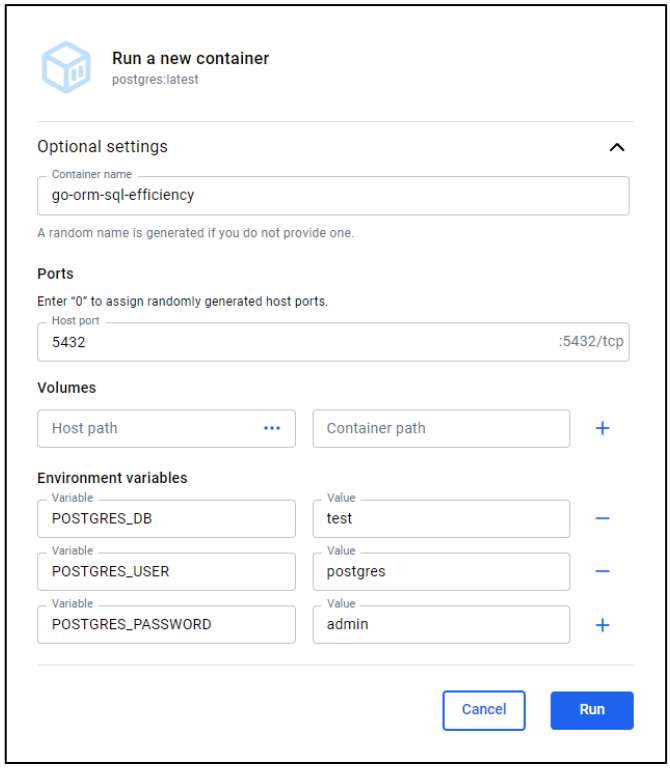

Рисунок 5.3 – Налаштування Docker-контейнера (рисунок виконаний самостійно)

Після запуску контейнера необхідно виконати єдиний SQL-скрипт міграції, який створює таблиці згідно з ER-моделлю та наповнює їх первинними даними.

5.3 Опис програмного підходу до проведення експериментів

Під час проведення експериментів необхідно отримувати п’ять метрик:

– середня затримка: вимірюється шляхом фіксування початкового часу перед виконанням операції та обчислення різниці після її завершення. Ця метрика показує, скільки в середньому триває одна операція, що важливо для оцінки загальної швидкодії системи. Вимірюється в мілісекундах та обраховується за наступною формулою 5.1:

$$AvgLatency = \frac{1}{N}\sum_{i=1}^{N}\frac{t_i}{10^6} \tag{5.1}$$

де N – це кількість операцій,

t_i – це час виконання i-ої операції;

– 95-й перцентиль затримки: обчислюється як значення затримки, нижче якого містяться 95% усіх індивідуальних затримок. Важливо для розуміння «довгого хвоста» повільних запитів, що може знижувати якість користувацького досвіду. Вимірюється в мілісекундах та обраховується за наступною формулою 5.2:

$$P95Latency = \frac{t_{[0,95N]}}{10^6} \tag{5.2}$$

де N – це кількість операцій,

$t_{(1)} \le t_{(2)} \le \cdots \le t_{(N)}$ – це впорядковані за зростанням затримки;

– пропускна здатність: розраховується як відношення кількості виконаних операцій до загального часу їхнього виконання. Ця метрика показує, скільки операцій система здатна обробити за одну секунду, що критично для оцінки її масштабованості під високим навантаженням. Вимірюється як операцій за секунду та обраховується за наступною формулою 5.3:

$$Throughput = \frac{N}{T_{total}} \quad (5.3)$$

де N – це кількість операцій,

T_{total} – це сумарний час виконання всіх операцій у секундах;

– середнє споживання оперативної пам’яті: вимірюється шляхом зчитування значень загального обсягу пам’яті до та після виконання операцій. Важлива для оцінки ефективності використання ресурсів і планування інфраструктури. Вимірюється в мегабайтах та обраховується за наступною формулою 5.4:

$$AvgRAM = \frac{\Delta M}{N \times 1024^2} \quad (5.4)$$

де N – це кількість операцій,

ΔM – це різниця в загальному обсязі пам’яті до та після виконання;

– час пауз GC: обчислюється як різниця в сумарному часі пауз GC до та після виконання операцій. Дає уявлення про вплив збірника сміття на загальні затримки та стабільність системи. Вимірюється в мілісекундах та обраховується за наступною формулою 5.5:

$$GCPause = \frac{\Delta P}{10^6} \quad (5.5)$$

де ΔP – це різниця в загальному часі пауз до та після виконання N операцій.

У Додатку Д представлено програмну реалізацію структур і функцій, які здійснюють збір та обчислення зазначених метрик.

5.4 Проведення експериментальних досліджень

Після написання додатка з трьома модулями переходимо до вимірювання обраних метрик: кожні з семи операцій, чотири прості та три складні, виконується послідовно 10 000 разів.

Результати вимірювань для операції створення представлені у таблицях 5.1.

Таблиця 5.1 – Результати виконання запитів на створення нового продукту (таблиця виконана самостійно)

Метрики / Альтернативи	GORM	Ent	database/sql
Середня затримка (мс)	2,5343	2,0303	2,4251
95-й перцентиль затримки (мс)	4,6652	2,6053	4,4141
Пропускна здатність (оп/сек)	394,5868	492,5464	412,3479
Середнє споживання ОЗП (Мб)	0,0068	0,0034	0,0006
Час пауз GC (мс)	2,2064	1,0053	0,0000

Результати вимірювань для операції отримання представлені у таблиці 5.2.

Таблиця 5.2 – Результати виконання запитів на отримання інформації клієнта за ідентифікатором (таблиця виконана самостійно)

Метрики / Альтернативи	GORM	Ent	database/sql
Середня затримка (мс)	0,3139	0,6479	0,6169
95-й перцентиль затримки (мс)	0,6498	0,9998	0,9981
Пропускна здатність (оп/сек)	3186,2296	1543,3845	1620,9478
Середнє споживання ОЗП (Мб)	0,0044	0,0040	0,0011
Час пауз GC (мс)	0,0000	0,5543	0,1128

Результати вимірювань для операції оновлення представлені у таблиці 5.3.

Таблиця 5.3 – Результати виконання запитів на оновлення ціни продукту за ідентифікатором (таблиця виконана самостійно)

Метрики / Альтернативи	GORM	Ent	database/sql
Середня затримка (мс)	2,2002	3,1357	1,9903
95-й перцентиль затримки (мс)	3,1580	3,7838	2,5135
Пропускна здатність (оп/сек)	454,5036	318,9031	502,4480
Середнє споживання ОЗП (Мб)	0,0062	0,0052	0,0003
Час пауз GC (мс)	3,0859	0,1206	0,0000

Результати вимірювань для операції видалення представлені у таблиці 5.4.

Таблиця 5.4 – Результати виконання запитів на видалення продукту за назвою (таблиця виконана самостійно)

Метрики / Альтернативи	GORM	Ent	database/sql
Середня затримка (мс)	3,6118	3,3223	3,3006
95-й перцентиль затримки (мс)	4,3545	4,0859	4,0921
Пропускна здатність (оп/сек)	276,8694	300,9072	302,9795
Середнє споживання ОЗП (Мб)	0,0052	0,0019	0,0003
Час пауз GC (мс)	3,6996	0,8759	0,0000

Результати вимірювань для транзакційної операції представлені у таблиці 5.5.

Таблиця 5.5 – Результати виконання запитів на створення замовлення з продуктами для конкретного клієнта за ідентифікатором (таблиця виконана самостійно)

Метрики / Альтернативи	GORM	Ent	database/sql
Середня затримка (мс)	2,5915	3,2192	3,1404
95-й перцентиль затримки (мс)	3,2037	3,9317	3,7949
Пропускна здатність (оп/сек)	385,8746	310,6328	318,4332
Середнє споживання ОЗП (Мб)	0,0188	0,0073	0,0019
Час пауз GC (мс)	3,4119	0,2680	0,0000

Результати вимірювань для агрегаційної операції представлені у таблиці 5.6.

Таблиця 5.6 – Результати виконання запитів на отримання статистики по замовленнях та витратах клієнта за ідентифікатором (таблиця виконана самостійно)

Метрики / Альтернативи	GORM	Ent	database/sql
Середня затримка (мс)	1,7731	2,2285	2,2091
95-й перцентиль затримки (мс)	2,4585	2,8527	2,8251
Пропускна здатність (оп/сек)	563,9747	448,7343	452,6629
Середнє споживання ОЗП (Мб)	0,0046	0,0008	0,0008
Час пауз GC (мс)	1,4302	0,8388	0,0000

Результати вимірювань для операції складного з’єднання представлені у таблиці 5.7.

Таблиця 5.7 – Результати виконання запитів на отримання інформації про продажі топ продуктів (таблиця виконана самостійно)

Метрики / Альтернативи	GORM	Ent	database/sql
Середня затримка (мс)	6,0444	6,1471	6,4545
95-й перцентиль затримки (мс)	6,8305	6,7861	7,3262
Пропускна здатність (оп/сек)	165,4425	162,6774	154,9314
Середнє споживання ОЗП (Мб)	0,0086	0,0013	0,0013
Час пауз GC (мс)	7,7937	0,5720	0,0000

За наведеними в таблицях даними буде здійснено подальший аналіз результатів експерименту.

5.5 Аналіз отриманих результатів

Для операцій створення: Ent показав найкращі результати за швидкістю та пропускною здатністю, споживаючи при цьому значно менше пам'яті (майже вдвічі) порівняно з GORM. Пакет database/sql був лише незначно (близько 10%) повільніший за Ent, але продемонстрував мінімальне споживання пам'яті та повну відсутність пауз GC. GORM відставав від Ent за швидкістю та мав найбільше споживання пам'яті.

Для операцій отримання: GORM виявився найшвидшим за середньою затримкою та пропускною здатністю, працюючи помітно швидше за Ent та database/sql. Ent і database/sql мали схожі показники затримки, проте Ent використовував значно більше пам'яті (приблизно в чотири рази), ніж database/sql. Пакет database/sql показав найкращу ефективність за споживанням пам'яті та відсутність пауз GC.

Для операцій оновлення: database/sql продемонстрував найкращу швидкість (бувши на 10% швидшим за GORM і на 60% швидшим за Ent) та найменше споживання пам'яті, з мінімальними паузами GC. GORM зайняв проміжну позицію за

затримкою, але споживав значно більше пам'яті (понад у 20 разів), ніж database/sql. Ent був найповільнішим і потребував помітно більше пам'яті (понад у 15 разів) за database/sql.

Для операцій видалення: database/sql та Ent працювали з майже однаковою швидкістю, випереджаючи GORM на 8-10%. За споживанням пам'яті database/sql залишався найощадливішим, Ent потребував приблизно у 6 разів більше, а GORM – понад у 15 разів більше пам'яті. Ent лідирував за 95-м перцентилем затримки.

Для транзакційних операцій: GORM показав найкращу швидкість (приблизно на 20% швидше) та найвищу пропускну здатність. Однак, він споживав майже в 10 разів більше пам'яті, ніж database/sql. database/sql був найефективнішим за споживанням пам'яті та повністю уникнув пауз GC, тоді як Ent показав оптимальний баланс за пам'яттю серед ORM.

Для агрегаційних операцій: GORM демонстрував найкращу швидкість (приблизно на 20% вище) та пропускну здатність (на 25% вище). При цьому GORM споживав у 5-6 разів більше пам'яті. Ent і database/sql показували майже однакові результати за швидкістю та були значно ефективнішими за пам'яттю; database/sql мав перевагу у відсутності пауз GC.

Для складних JOIN-запитів: GORM і Ent показували майже однакову середню затримку, випереджаючи database/sql на 5-7%. За пропускною здатністю GORM трохи випереджав інших. Ent і database/sql залишалися найефективнішими за пам'яттю, тоді як GORM споживав у 6-7 разів більше пам'яті. Ent лідирував за 95-м перцентилем затримки.

Загалом, кожен з цих програмних підходів має свої переваги та недоліки, і вибір залежить від конкретного завдання та пріоритетів розробника. Однак, на основі проведеного дослідження можна запропонувати такі обґрунтовані рекомендації:

– database/sql є оптимальним вибором, якщо необхідно максимально оптимізувати швидкість виконання запитів та мінімізувати витрати пам’яті та пауз GC, особливо для операцій оновлення та видалення. Він демонструє

видатну ефективність ресурсів у всіх сценаріях, лише трохи поступаючись ORM у швидкості для складних запитів, але значно випереджаючи їх за економією пам'яті;

- GORM забезпечує найвищу швидкість у більшості операцій читання. Його варто використовувати, коли пріоритетом є критично низька затримка читання та висока пропускна здатність за будь-яку ціну ресурсів, оскільки він демонструє значно більше споживання пам’яті та помітніші паузи GC;
- Ent є компромісним рішенням, якщо потрібен баланс між продуктивністю та економним використанням пам’яті, а також для ефективного створення даних. У більшості операцій його швидкість лише на 5-20% гірша, ніж у GORM, але споживання пам’яті на порядок менше, а паузи GC значно коротші;
- якщо є жорсткі обмеження на обсяг пам’яті, слід обережно використовувати GORM, особливо для асоціативних запитів, де він демонструє значно більше споживання пам’яті порівняно з database/sql та Ent;
- у проєктах з великою кількістю операцій читання та створення даних, GORM і Ent обидва демонструють високі результати, але database/sql буде вигіднішим, якщо потрібно знизити споживання пам’яті та уникнути пауз GC.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було проведено аналіз предметної галузі, у межах якого описано підходи до доступу до баз даних у Go-додатках: порівняно ORM-бібліотеки та прямий SQL-підхід, їхні переваги та недоліки. Крім того, було здійснено огляд й аналіз літературно-наукових джерел, які досліджують порівняльні характеристики ORM та SQL підходів у контексті продуктивності та ресурсомісткості, що дозволило визначити актуальні проблеми й тренди в цій галузі.

Було виконано багатокритеріальний аналіз ORM-бібліотек за такими критеріями: документація та навчальні матеріали, кількість підтримуваних СУБД, популярність серед розробників на GitHub, легкість інтеграції та використання, та підтримка спільноти та оновлення. За результатами цього аналізу визначено Парето-оптимальні альтернативи – GORM та Ent – як найперспективніші.

Було розроблено експериментальний додаток з трьома незалежними модулями: один на базі GORM, другий – на базі Ent, і третій – з database/sql. Спроєктовано логічну модель бази даних для предметної області електронна комерція, реалізовано схему у PostgreSQL через Docker і автоматизовано міграції за допомогою golang-migrate.

Було проведено серію експериментальних досліджень для семи типів операцій з базою даних, під час яких вимірювалися: середня затримка, 95-й перцентиль затримки, пропускна здатність, середнє споживання оперативної пам'яті та час пауз GC. Отримані дані свідчать, що немає єдиного універсально ефективного підходу. Пакет database/sql показав найвищу ефективність ресурсів та оптимальний результат для операцій оновлення та видалення. Бібліотека Ent демонструє найкращу продуктивність для створення і є збалансованим вибором. GORM забезпечує максимальну зручність розробки та найкращі показники для складних операцій читання, проте з вищим споживанням пам'яті. Тому вибір залежить від пріоритету:

database/sql для критичних сценаріїв з обмеженими ресурсами, Ent для збалансованих рішень, або GORM для швидкого прототипування та складних читань.

За результатами кваліфікаційної роботи були створені тези доповіді, які було представлено на молодіжній конференції MIT@AISm-2025, яка проводилася в межах 1-ї Міжнародної науково-практичної конференції «СУЧАСНІ ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ТА СИСТЕМИ ШТУЧНОГО ІНТЕЛЕКТУ MIT@AIS-2025».

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Sql package - database/sql - Go Packages. *Go Packages - Go Packages*. URL: https://pkg.go.dev/database/sql (дата звернення: 17.04.2025).

2. Gorm. *GORM*. URL: https://gorm.io/index.html (дата звернення: 17.04.2025).

3. XORM - eXtra ORM for Go. *XORM*. URL: https://xorm.io/ (дата звернення: 18.04.2025).

4. Ent. *ent*. URL: https://entgo.io/ (дата звернення: 18.04.2025).

5. GitHub - volatiletech/sqlboiler: generate a go ORM tailored to your database schema. *GitHub*. URL: https://github.com/volatiletech/sqlboiler (дата звернення: 19.04.2025).

6. Procaccianti G., Lago P., Diesveld W. Energy Efficiency of ORM Approaches: an Empirical Evaluation // Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. 2016. URL: https://doi.org/10.1145/2961111.2962586 (дата звернення: 20.04.2025).

7. Colley D., Stanier C., Asaduzzaman M. The Impact of Object-Relational Mapping Frameworks on Relational Query Performance // 2018 International Conference on Computing, Electronics & Communications Engineering (iCCECE). 2018. С. 47–52. URL: https://doi.org/10.1109/ICCECOME.2018.8659222 (дата звернення: 21.04.2025).

8. Chen T., Shang W., Jiang Z., Hassan A., Nasser M. N., Flora P. Finding and Evaluating the Performance Impact of Redundant Data Access for Applications that are Developed Using Object-Relational Mapping Frameworks // IEEE Transactions on Software Engineering. 2016. Vol. 42. P. 1148–1161. URL: https://doi.org/10.1109/TSE.2016.2553039 (дата звернення: 22.04.2025).

9. Zmaranda D., Pop-Fele L.-L., Gyorodi C., Gyorodi R., Pecherle G. Performance Comparison of CRUD Methods using NET Object Relational Mappers: A Case Study // International Journal of Advanced Computer Science and Applications. 2020. Vol. 11. URL: https://doi.org/10.14569/ijacsa.2020.0110107 (дата звернення: 23.04.2025).

10. Мазурова О. О., Андрущенко М. О., Широкопетлєва М. С. Дослідження методів програмної реалізації Cosmos DB API на платформі .NET // Сучасний стан наукових досліджень та технологій в промисловості. 2023. № 2 (24). С. 118–130. URL: https://openarchive.nure.ua/handle/document/25421 (дата звернення: 25.04.2025).

11. Широкопетлєва М. С., Черепанова Ю. Ю. Проєктування програмної системи тестування знань з мови SQL // Materialy XII Miedzynarodowej naukowi-praktycznej konferencji «Dynamika naukowych badan-2016», 07–15 lipca 2016 roku. Volume 6. Przemyśl: Nauka i studia. С. 32–35. URL: http://openarchive.nure.ua/handle/document/5677 (дата звернення: 29.04.2025).

12. PostgreSQL. *PostgreSQL*. URL: https://www.postgresql.org/ (дата звернення: 29.04.2025).

13. Technology | 2024 stack overflow developer survey. *Stack Overflow Insights - Developer Hiring, Marketing, and User Research*. URL: https://survey.stackoverflow.co/2024/technology#1-databases (дата звернення: 30.04.2025).

14. Documentation. *Docker*. URL: https://docs.docker.com/ (дата звернення: 30.04.2025).

15. GitHub - golang-migrate/migrate: database migrations. CLI and golang library. *GitHub*. URL: https://github.com/golang-migrate/migrate (дата звернення: 01.05.2025).

16. GitHub - yahn1ukov/go-orm-sql-efficiency. *GitHub*. URL: https://github.com/yahn1ukov/go-orm-sql-efficiency (дата звернення: 09.06.2025).

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ ЗА НАУКОВИМИ НАПРЯМАМИ КЕРІВНИКА ТА НАУКОВЦІВ КАФЕДРИ ПРОГРАМНОЇ ІНЖЕНЕРІЇ

10. Мазурова О. О., Андрущенко М. О., Широкопетлєва М. С. Дослідження методів програмної реалізації Cosmos DB API на платформі .NET // Сучасний стан наукових досліджень та технологій в промисловості. 2023. № 2 (24). С. 118–130. URL: https://openarchive.nure.ua/handle/document/25421 (дата звернення: 25.04.2025).

11. Широкопетлєва М. С., Черепанова Ю. Ю. Проєктування програмної системи тестування знань з мови SQL // Materialy XII Miedzynarodowej naukowi-praktycznej konferencji «Dynamika naukowych badan-2016», 07–15 lipca 2016 roku. Volume 6. Przemyśl: Nauka i studia. С. 32–35. URL: http://openarchive.nure.ua/handle/document/5677 (дата звернення: 29.04.2025).