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 залежності продуктивності реляційних та графових баз даних від даних
(тема)

Виконав:
студент 2 курсу, групи ІПЗм-22-1

Недаєв І.С.
(прізвище, ініціали)

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

Керівник доц. Мазурова О.О.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ З.В.Дудар
(підпис) (прізвище, ініціали)

2024 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)
«___» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Недаєву Ігорю Сергійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи «Дослідження залежності продуктивності реляційних та графових баз даних від даних»
Затверджена наказом по університету від 29.03. 2024р. № 250 Ст
2. Термін подання студентом роботи до екзаменаційної комісії 17.06.2024
3. Вихідні дані до роботи опис та порівняння досліджуваних реляційної та графових БД, вимоги до розробки схеми бази даних для проведення досліджень за обраною предметною областю, програмна реалізація для проведення дослідження для обраних БД, аналіз та структурування отриманих результатів дослідження, мова програмування C#, технології .NET 6.0, СКБД Microsoft SQL Server, СКБД Neo4j, середовище розробки Visual Studio.
4. Перелік питань, що потрібно опрацювати в роботі
вступ, аналіз предметної області, вибір предметної області для побудови схеми баз даних для проведення дослідження, визначення параметрів порівняння досліджуваних БД, опис програмних реалізацій та дослідження обраних БД, аналіз та структурування отриманих результатів.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі та постановка задачі	23.01 – 05.02.24	*виконано*
2	Аналіз та вибір СКБД для дослідження	05.02 – 20.02.24	*виконано*
3	Аналіз та моделювання предметної області	20.02 – 01.03.24	*виконано*
4	Планування експериментів	01.03 – 10.03.24	*виконано*
5	Програмна реалізація досліджуваних СКБД	10.03 – 20.03.24	*виконано*
6	Експериментальні дослідження	20.03 – 20.04.24	*виконано*
7	Аналіз результатів експериментальних досліджень та розробка рекомендацій	20.04 – 01.05.24	*виконано*
8	Написання та оформлення статті та тез доповіді	01.05 – 10.05.24	*виконано*
9	Підготовка пояснювальної записки	10.05 – 31.05.24	*виконано*
10	Підготовка презентації та доповіді	31.05 – 05.06.24	*виконано*
11	Нормоконтроль	05.06 – 08.06.24	*виконано*
12	Рецензування	08.06 – 12.06.24	*виконано*
13	Занесення диплома в електронний архів	12.05.2024	*виконано*
14	Попередній захист	13.05.2024	*виконано*
15	Допуск до захисту у зав. кафедри	15.06.2024	*виконано*

Дата видачі завдання 23 січня 2024р.

Студент (ка) _____ Недаєв І.С.
(підпис)

Керівники роботи _____ доц. Мазурова О.О.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка містить: 64 с., 6 рис., 11 табл., 18 джерел.

ГРАФОВА БАЗА ДАНИХ, ЗАПИТ, МЕТРИКА, РЕЛЯЦІЙНА БАЗА ДАНИХ, MS SQL, .NET, NEO4J.

Об'єктом дослідження є процес роботи з графовими та реляційними базами даних, а предметом дослідження є методи зберігання, обробки та доступу до даних в системах керування реляційними та графовими базами даних.

Метою даної роботи є формулювання більш ефективних рекомендацій стосовно вибору моделі даних та відповідної СКБД з урахуванням контексту даних, що будуть покладені в основу відповідних баз даних, на базі аналізу результатів їх експериментальних досліджень.

Методи розробки базуються на платформі .NET, мові програмування C#, реляційній СКБД MS SQL та графовій СКБД Neo4j.

У результаті роботи було сформовано рекомендації стосовно вибору моделі даних та відповідної СКБД з урахуванням контексту даних, що будуть покладені в основу відповідних баз даних, на базі аналізу результатів їх експериментальних досліджень.

GRAPH DATABASE, QUERY, METRIC, RELATIONAL DATABASE, MS SQL, .NET, NEO4J.

The object of research is the process of working with graph and relational databases, and the subject of research is the methods of storing, processing and accessing data in relational and graph database management systems.

The purpose of this work is to formulate more effective recommendations for choosing a data model and an appropriate DBMS, taking into account the context of the data that will form the basis of the respective databases, based on the analysis of the results of their experimental studies.

The development methods are based on the .NET platform, C# programming language, MS SQL relational DBMS and Neo4j graph DBMS.

As a result of the work, based on the analysis of the results of experimental studies, recommendations for choosing a data model and an appropriate DBMS were made, taking into account the context of the data that will form the basis of the respective databases.

Заява щодо самостійного виконання кваліфікаційної роботи та можливості її публікації в електронному архіві відкритого доступу ElArKhNURE.

Я, Недаєв Ігор Сергійович, студент гр. ІПЗм-22-1, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Дослідження залежності продуктивності реляційних та графових баз даних від даних», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElArKhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ВСТУП

На сьогоднішній день бази даних є невід’ємною частиною більшості прикладних програмних систем. Під час розробки використовуються різні типи баз даних, кожен із яких має своє призначення, сильні та слабкі сторони. Системи керування базами даних (СКБД) кожного типу мають свою галузь використання та були створені для вирішення різних типів завдань. Для моделювання складних та слабко структурованих предметних галузей, в яких наявні різні типи даних та зв’язків, перед розробниками постає важлива задача визначити, який тип СКБД краще підходить у даному контексті.

У даній роботі досліджується продуктивність роботи реляційних та графових баз даних на основі вимірюваних метрик під час виконання запитів та їх залежність від даних.

Об'єктом дослідження є процес роботи з графовими та реляційними базами даних, а предметом дослідження є методи зберігання, обробки та доступу до даних в системах керування реляційними та графовими базами даних.

Метою даної роботи є формулювання більш ефективних рекомендацій стосовно вибору моделі даних та відповідної СКБД з урахуванням контексту даних, що будуть покладені в основу відповідних баз даних, на базі аналізу результатів їх експериментальних досліджень.

Під час виконання кваліфікаційної роботи був проведений аналіз проблемної галузі продуктивності роботи графових та реляційних СКБД на основі публікацій світових та вітчизняних фахівців у проблемній області продуктивності NoSQL СКБД у порівнянні із реляційними (див. додаток А). Темою роботи обрано дослідження впливу типів даних та їх зв’язності на продуктивність реляційних та графових СКБД.

Було спроектовано процес проведення експерименту дослідження, що включає в себе вибір предметної галузі, проектування схем та таблиць БД, проектування запитів для проведення дослідження, вибір метрик, планування умов проведення експерименту, розробка проектів для роботи із досліджуваними СКБД, розробка допоміжного програмного забезпечення.

Кваліфікаційна робота пройшла успішну перевірку на академічну доброчесність (див. додаток Б),

На основі плану проведення дослідження та його результатів було розроблено презентацію (див. додаток В).

За результатами дослідження опубліковано тези «Investigation of the Efficiency Dependence of Relational and Graph Databases on Data» на 12-тій Міжнародній науковій та технічній конференції «Інформаційні системи та технології» (IST-2023) (див. додаток Г).

Робота пройшла перевірку на відповідність оформлення вимогам ДСТУ 3008: 2015 (див. додаток Д).

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі дослідження

У сучасних програмних продуктах виникає потреба не лише у зберіганні текстової або числової інформації, але й у зберіганні зображень, відео та аудіо файлів. Зокрема, у зв'язку із розвитком технологій та збільшенням обсягу даних, з якими працюють програмні продукти, все частіше виникає необхідність зберігання їх у великому обсязі (Big Data). Це зумовлює використання різних типів БД, кожен із яких має своє призначення, переваги та недоліки.

Різні системи керування базами даних (СКБД) використовують свої механізми зберігання та обробки даних. Реляційні бази даних вважаються класичним підходом до збереження даних, вони зберігають інформацію у вигляді таблиць та зв'язків між ними [1].

Реляційні СКБД для свого функціонування потребують приведення інформації до відповідного формату та структури. Вони використовують такі основні типи даних [2]:

- текстові (стандартні символи, рядки та блоки тексту);
- числові (цілі числа, числа з плаваючою комою);
- дата та час (дата, час, часові проміжки);
- булеві дані (логічні значення true/false);
- двійкові (файли, зображення тощо).

Для обробки та збереження великих обсягів неструктурованих даних використовують NoSQL системи. На відміну від традиційних реляційних БД, що використовують таблиці із заздалегідь визначеними схемами, системи NoSQL використовують гнучкі моделі, що можуть адаптуватися до змін у структурі даних та здатні до горизонтального масштабування [3-5]. На відміну від реляційних, NoSQL СКБД використовують різноманітні формати даних, що не потребують чіткої структури [3].

Також серед NoSQL систем слід виділити підтип графових БД, що спеціалізуються на збереженні та обробці даних зі складними зв'язками. Вони

використовуються у випадках, коли важливо зберігати та швидко обробляти інформацію про відносини між різними об'єктами. Наприклад, графові бази даних особливо підходять для вирішення задач, подібних до соціальних мереж, де важливо відстежувати та аналізувати зв'язки між користувачами, або рекомендаційних систем, де важливо виявляти схожість та зв'язки між продуктами та користувачами [6].

Існує декілька підходів до зберігання цифрової інформації [7], серед них:

– файлові системи;
– бази даних;
– хмарні сховища (cloud storage) – використання віддалених серверів, доступних через Інтернет.

Дане дослідження зосереджене на використанні БД для обробки та зберігання цифрових даних. У свою чергу БД може зберігати двійкові дані у вигляді рядків BASE64, масивів байтів або об’єктів BLOB. Також реляційні СКБД містять тип даних Image, що також спрямований на вирішення даної проблеми. Для проведення дослідження було вирішено обрати формат збереження двійкових даних у виді масивів байтів.

Оскільки різні СКБД мають різний підхід до взаємодії із даними, актуальним є питання оптимального використання різних типів БД для різних типів вихідних даних. Вибір між реляційними та графовими базами даних залежить від характеристики даних та вимог до продуктивності системи.

У літературі існують деякі загальні рекомендації щодо вибору бази даних, але чіткого розподілу немає – вибір залишається за розробниками. Багато досліджень показують, що графові бази даних набагато ефективніші за реляційні при використанні простих запитів [8], тоді як інші дослідження вказують на те, що зв’язність даних несуттєво впливає на продуктивність графових СКБД, проте має значний вплив на продуктивність реляційних СКБД [9, 10].

У ході дослідження потрібно дослідити роботу реляційних та графових СКБД на прикладі основних сценаріїв використання. Після визначення чинників, що впливають на ефективність роботи зазначених СКБД, у результаті повинні бути

сформовані рекомендації щодо ситуацій у одна із зазначених систем проявляє себе краще за іншу.

Для більш точних результатів дослідження обох типів БД має проводитись на одних даних із використанням однакових запитів. Проте через різні архітектури та структури даних доведеться застосовувати різні підходи до проектування БД та запитів, проте результати їх виконання мають бути однаковими.

Для проведення дослідження має бути обрана така предметна область, щоб вона мала великий обсяг різноманітних даних, мала велику кількість задач та щоб мала як прості зв’язки між великою кількістю даних – так і складні зв’язки між невеликою кількістю даних.

Для обраної предметної області потрібно створити реляційну та графові БД для однакового набору даних, розробити запити, що будуть надавати однаковий результат.

1.2 Постановка задачі

Об'єктом даного дослідження є процес роботи з графовими та реляційними базами даних. Це передбачає вивчення способів зберігання, обробки та доступу до даних у цих системах, а також їх відмінностей з точки зору продуктивності.

Предметом цього дослідження є методи зберігання, обробки та доступу до даних у реляційних та графових системах керування базами даних.

У результаті дослідження мають бути отримані сценарії використання графових баз даних та сценарії використання реляційних баз даних, а також переваги та недоліки графових та реляційних баз даних в залежності від типів та зв’язності вхідних даних.

Для досягнення мети дослідження необхідно розробити програмну систему для тестування продуктивності різних систем керування базами даних: бібліотеку для проведення замірів досліджуваних метрик та проекти з реалізаціями графових (Neo4j) та реляційних СКБД (Entity Framework та Microsoft SQL Server), а також розробити відповідні бази даних із різною структурою зв'язків та забезпечити їх належне функціонування.

Завдання дослідження:

- провести аналіз наявних реляційних та графових СКБД та обрати ті, що найбільше підходять для даного дослідження;
- обрати предметну область, що задовольняють потребам дослідження, та провести її аналіз та моделювання (побудувати інфологічну модель, реляційну та графову логічні моделі);
- розробити план експериментального дослідження (визначити умови проведення експериментів, метрики для дослідження, розробити запити для досліджуваних даних);
- спроектувати та розробити програмне забезпечення для проведення дослідження;
- провести експериментальне дослідження продуктивності роботи зазначених СКБД, за його результатами сформувати рекомендації щодо використання досліджуваних СКБД із різними даними (тип даних, їх обсяг, їх зв'язність тощо).

2 ОПИС ПРИЙНЯТИХ ПРОЄКТНИХ РІШЕНЬ

2.1 Аналіз та вибір реляційних та графових СКБД для дослідження

Для проведення дослідження потрібно обрати одну або дві графові СКБД. Щоб дослідження було змістовним, вибір слід здійснити у ході порівняння СКБД між собою у контексті дослідження, що проводиться. Загальні рекомендації щодо обраних СКБД можуть бути неактуальними під час даного дослідження, саме тому їх потрібно дослідити, спираючись на специфіку написання магістерскької роботи.

Neo4j – провідна система управління графовими базами даних, призначена для зберігання та здійснення запитів до сильно зв'язаних даних [11]. Серед основних особливостей:

- нативна графова база даних: створена спеціально для графових моделей даних, пропонує ефективне зберігання та обхід зв'язків;
- cypher Query Language: потужна та виразна мова запитів, пристосована для роботи з графами;
- відповідність стандарту ACID: забезпечує цілісність даних завдяки підтримці атомарності, узгодженості, ізоляції та довговічності;
- масштабованість: підтримує кластеризацію та горизонтальне масштабування для роботи з великими наборами даних;
- графові алгоритми: надає бібліотеку графічних алгоритмів для поглибленого аналізу;
- візуалізація: вбудовані інструменти для дослідження та візуалізації графічних структур.

ArangoDB – це багатомодельна база даних, яка поєднує ключ-значення, документну та графову моделі даних в межах однієї системи [11, 12]. Серед основних особливостей:

- багатомодельність: підтримує моделі даних "ключ-значення", "документ" та "граф" в єдиній базі даних;
- мова запитів AQL: пропонує універсальну мову запитів для пошуку та маніпулювання даними;

- ACID-транзакції: забезпечує надійну узгодженість даних завдяки підтримці транзакцій;
- вбудовані графічні можливості: дозволяє ефективно зберігати графічні дані та робити запити до них;
- Foxx Microservices: дозволяє розробляти безсерверні JavaScript мікросервіси у базі даних.

OrientDB – це багатомодельна NoSQL база даних, яка підтримує документні, графові та моделі даних типу "ключ-значення" [13]. Серед основних особливостей:

- багатомодельність: поєднує декілька моделей даних, включаючи документу та графову, в одній базі даних;
- SQL-подібна мова запитів: надає розроблену на основі SQL мову запитів для роботи із даними;
- ACID-транзакції: забезпечує узгодженість даних завдяки підтримці транзакцій;
- розподілена і масштабована: підтримує горизонтальне масштабування та Multi-master реплікацію;
- геопросторові можливості: включає геопросторове індексування та запити до даних на основі місцезнаходження.

Amazon Neptune – це повністю керований сервіс графових баз даних від AWS, призначений для створення додатків з сильно зв'язаними даними [11]. Серед основних особливостей:

- повна керованість: пропонує автоматизоване резервне копіювання, масштабування та обслуговування БД, зменшуючи операційні накладні витрати;
- сумісність: підтримує мови запитів Gremlin (Apache TinkerPop) та SPARQL;
- висока доступність: забезпечує реплікацію даних та відмовостійкість для підвищення доступності;
- безпека: забезпечує шифрування в стані спокою та під час передачі, а також інтеграцію з IAM для контролю доступу;

– інтеграція: легко інтегрується з іншими сервісами AWS, такими як Lambda, S3 та CloudWatch.

Azure Cosmos DB – це глобально розподілена багатомодельна служба баз даних, що надається Microsoft Azure, яка підтримує документну, графову, "ключ-значення" та модель даних "сімейство стовпців" [14]. Серед основних особливостей:

– глобальне розподілення: дозволяє реплікувати дані між регіонами Azure для доступу з низькою затримкою;
– багатомодельність: підтримує різні моделі даних, зокрема SQL, MongoDB, Gremlin, Cassandra та Azure Table Storage;
– підтримка SLA: забезпечує високу доступність з гарантованим часом безвідмовної роботи та угодами про рівень обслуговування з низькою затримкою;
– масштабованість: масштабування пропускної здатності та сховища відбувається незалежно, що робить його придатним для різних робочих навантажень;
– безпека: надає надійні функції безпеки, включаючи шифрування, автентифікацію та контроль доступу на основі ролей.

Критерії дослідження:

– продуктивність (Performance) – продуктивність оцінює, наскільки добре база даних може виконувати запити до графів і ефективно обходити зв'язки. Вона включає в себе вимірювання часу виконання запитів, пропускної здатності та часу відгуку;
– масштабованість (Scalability) – масштабованість оцінює здатність бази даних обробляти зростаючі обсяги даних і збільшені робочі навантаження. Вона включає оцінку горизонтального масштабування, балансування навантаження і механізмів розподілу даних;
– підтримка графічної моделі даних – підтримка графових моделей даних враховує, наскільки добре база даних обслуговує специфічні для графів структури даних, включаючи вузли, зв'язки і властивості;

– узгодженість даних та відповідність ACID – узгодженість даних оцінює, чи підтримує база даних цілісність даних і чи дотримується вона властивостей ACID (атомарність, узгодженість, ізольованість, довговічність). Вона оцінює, наскільки добре здійснюється управління транзакціями;
– простота використання та розробки – простота використання та розробки враховує зручність бази даних, доступність інструментів для розробників та силу спільноти користувачів. Він також включає якість документації та ресурсів.

Опис шкали оцінок за критеріями:

а) час відгуку (Response time) – інтервальна шкала (дані відносні, були взяті з декількох джерел, включаючи офіційну документацію, але їх значення розбігаються, тому числа приблизні);
   1) ~10 мілісекунд (мс): відображає швидкий час відгуку, що вказує на відмінну продуктивність;
   2) ~100 мс: означає хорошу продуктивність з досить швидким виконанням запитів;
   3) ~200 мс: вказує на прийнятну продуктивність, але з деякими можливостями для покращення.

б) масштабованість (Scalability) – порядкова шкала;
   1) відмінно: база даних може легко горизонтально масштабуватися та ефективно розподіляти робочі навантаження – 5 балів;
   2) добре: вона може масштабуватися з незначними проблемами і пропонує розумне балансування навантаження – 4 бали;
   3) задовільно: масштабованість обмежена, і потрібні значні зусилля, щоб впоратися з ростом. – 3 бали.

в) підтримка графічної моделі даних – порядкова шкала;
   1) графо-орієнтована: база даних спеціалізована для роботи з графічними даними, з відмінною підтримкою та оптимізацією – 10 балів;

2) багатомодельна: база даних підтримує кілька моделей даних, включаючи графіки, але може бути не такою оптимізованою – 5 балів;
3) не-графоцентрична: база даних в основному орієнтована на інші моделі даних, з обмеженою підтримкою графів – 1 бал.

г) узгодженість даних та відповідність ACID – порядкова шкала;
1) повна відповідність: база даних повністю відповідає властивостям ACID, забезпечуючи високу узгодженість даних – 5 балів;
2) часткова відповідність: база даних частково відповідає ACID, забезпечуючи певний рівень узгодженості даних – 3 бали;
3) обмежена відповідність: база даних має обмежену відповідність ACID, що потенційно загрожує узгодженості даних – 1 бал.

д) простота використання та розробки – порядкова шкала;
1) відмінно: база даних дуже зручна для користувача, пропонує широкі ресурси для розробників і має сильну спільноту – 10 балів.
2) добре: база даних забезпечує хороший користувацький досвід, інструменти для розробників і гідну підтримку спільноти – 8 балів.
3) задовільно: зручність використання, ресурси для розробників та підтримка спільноти дещо недостатні – 5 балів.

Оскільки для вибору альтернатив розглядались одні з найкращих графових БД – оцінки «Незадовільно» та «Погано» були опущені із переліку.

Після визначення альтернатив, критеріїв вибору та виміру відповідних даних була побудована таблиця порівняння графових баз даних (див. табл. 2.1).

Таблиця 2.1 – Графові бази даних зі значеннями критеріїв (створено самостійно)

Метрики /СКБД	Час відгуку (мс)	Масшта-бованість	Підтримка граф. моделі	Відповідність ACID	Простота використання та розробки
Neo4j	15	Добре	Графо-орієнтована	Повна відповідність	Відмінно
ArangoDB	40	Добре	Багато-модельна	Повна відповідність	Добре

Кінець таблиці 2.1

Метрики /СКБД	Час відгуку (мс)	Масшта-бованість	Підтримка граф. моделі	Відповідність ACID	Простота використання та розробки
OrientDB	45	Добре	Багато-модельна	Повна відповідність	Добре
Amazon Neptune	20	Відмінно	Графо-орієнтована	Повна відповідність	Добре
Azure Cosmos DB	20	Відмінно	Багато-модельна	Повна відповідність	Задовільно

Таблиця з переведеними до числового формату критеріями наведена нижче (див. табл. 2.2).

Таблиця 2.2 – Графові бази даних із числовими значеннями критеріїв (створено самостійно)

Метрики /СКБД	Час відгуку (мс)	Масшта-бованість	Підтримка граф. моделі	Відповідність ACID	Простота використання та розробки
Neo4j	15	4	10	5	10
ArangoDB	40	4	5	5	8
OrientDB	45	4	5	5	8
Amazon Neptune	20	5	10	5	8
Azure Cosmos DB	20	5	5	5	5

Оскільки критерій «Час відгуку» кращий, коли значення менше, а інші 4 критерії – навпаки кращі, коли їх значення більші, то критерій «Час відгуку» потрібно привести до принципу оптимальності «за максимумом».

Відповідно, отримаємо критерій «Зекономлений час» Оскільки менша кількість мс краща, то точкою відліку потрібно обрати найбільший час відгуку (45 мс) та вказати модуль його різниці із іншими альтернативами. Таким чином чим менший час відгуку – тим більшим буде число після оптимізації. Таблиця альтернатив після перетворення наведена нижче (див. табл. 2.3).

Таблиця 2.3 – Приведені до одного напряму значення (створено самостійно)

Метрики /СКБД	Зекономлений час (мс)	Масштабованість	Підтримка граф. моделі	Відповідність ACID	Простота використання та розробки
Neo4j	30	4	10	5	10
ArangoDB	5	4	5	5	8
OrientDB	0	4	5	5	8
Amazon Neptune	25	5	10	5	8
Azure Cosmos DB	25	5	5	5	5

При використанні принципу Парето на множині альтернатив 3 з них виявились гіршими при порівнянні (див. табл. 2.4, 2.5). Neo4j краща за ArangoDB та OrientDB по 3-ом параметрам та рівносильна по 2-ом; Amazon Neptune кращий за Azure Cosmos DB по 2-ом параметрам та рівносильна за 3-ома.

Таблиця 2.4 – Використання принципу Парето (створено самостійно)

Метрики /СКБД	Зекономлений час (мс)	Масштабованість	Підтримка граф. моделі	Відповідність ACID	Простота використання та розробки
Neo4j	30	4	10	5	10
~~ArangoDB~~	~~5~~	~~4~~	~~5~~	~~5~~	~~8~~
~~OrientDB~~	~~0~~	~~4~~	~~5~~	~~5~~	~~8~~
Amazon Neptune	25	5	10	5	8
~~Azure Cosmos DB~~	~~25~~	~~5~~	~~5~~	~~5~~	~~5~~

Таблиця 2.5 – Вигляд таблиці після використання принципу Парето (створено самостійно)

Метрики /СКБД	Зекономлений час (мс)	Масштабованість	Підтримка граф. моделі	Відповідність ACID	Простота використання та розробки
Neo4j	5	4	10	5	10
Amazon Neptune	0	5	10	5	8

Наступним кроком є нормування значень критеріїв з урахуванням ***min*** та ***max*** значень критеріїв, щоб привести їх до шкали [0; 1]. Для цього була використана формула 2.1:

$$f = \frac{f_{\text{вимір}} - f_{\boldsymbol{min}}}{f_{\boldsymbol{max}} - f_{\boldsymbol{min}}} \tag{2.1}$$

де $f_{\text{вимір}}$ – значення критерію,

$f_{\boldsymbol{min}}$– мінімальне значення критерію у стовпці,

$f_{\boldsymbol{max}}$– максимальне значення критерію у стовпці.

Нормована таблиця після застосування формули 2.1 до значень критеріїв наведена нижче (див. табл 2.6).

Таблиця 2.6 – Нормовані значення критеріїв (створено самостійно)

Метрики /СКБД	Зекономлений час (мс)	Масштабованість	Підтримка граф. моделі	Відповідність ACID	Простота використання та розробки
Neo4j	1	0	1	1	1
Amazon Neptune	0	1	1	1	0

Для вирішення багатокритеріальної задачі вибору доцільно використовувати лінійну адитивну згортку із ваговими коефіцієнтами, оскільки деякі критерії є важливішими за інші. Для формування вагових коефіцієнтів було використано пропорційний метод: «Зекономлений час» та «Підтримка графової моделі» у 3 рази важливіші за «Масштабованість» та «Відповідність ACID», оскільки безпосередньо для даного дослідження швидкість виконання запитів має найбільше значення, а швидкість виконання запитів залежить крім часу відгуку СКБД ще й від її оптимізації; та у 1,5 рази важливіші за «Простоту використання та розробки», від якої залежить якість дослідження та витрачений на нього час.

Розрахунок лінійної адитивної згортки із ваговими коефіцієнтами було проведено за формулою 2.2, після чого результати розрахунків були внесені до таблиці 2.7:

$$Z^* = \max_{i=1,m} \sum_{j=1}^{n} \alpha_j \beta_j a_{ij} \quad (2.2)$$

де α_j – нормуючі множники,

β_j– вагові коефіцієнти,

a_{ij} – значення критерію із таблиці.

Таблиця 2.7 – Розрахунок лінійної адитивної згортки із ваговими коефіцієнтами із використанням принципу Парето (створено самостійно)

Метрики /СКБД	Продук-тивність (мс)	Масшта-бованість	Підтримка граф. моделі	Відповід-ність ACID	Простота використання та розробки	Z^*
Neo4j	1	0	1	1	1	0,7
Amazon Neptune	0	1	1	1	0	0,3
Вагові коеф.	0,3	0,1	0,3	0,1	0,2	

За значенням функції адитивної згортки (останній стовпець таблиці 2.7) найкращою виявились альтернатива Neo4j, що має значення вище за Amazon Neptune. За результатами вирішення багатокритеріальної задачі вибору СКБД для дослідження було обрано альтернативу Neo4j.

Також потрібно було обрати реляційну СКБД для дослідження. Однією із найвідоміших та широко використовуваних є Microsoft SQL Server. Дана СКБД постійно отримує оновлення і є однією із провідних на даний час. Обрана СКБД має доволі легку інтеграцію, що зменшить час та складність розробки.

2.2 Вибір та аналіз предметної галузі для дослідження

Предметна галузь дослідження має бути обрана таким чином, щоб під час його проведення вирішувались проблеми із даними різних типів, різної зв’язності та різного обсягу. Здебільшого слід приділити увагу наявності цифрових даних, оскільки вони найбільше вирізняються серед досліджуваних типів.

Прикладом такої предметної області є футбольна статистика, де присутні як слабко зв’язані, так і сильно зв’язані дані, та де виникає потреба у використанні різноманітних типів даних. Прикладом слабко зв’язаних даних може бути історія цін гравця, що має зв’язки лише з конкретним гравцем, тоді як гравець має зв’язки із клубами, матчами, гольовими подіями тощо.

Сучасна футбольна статистика використовує багато типів даних, окрім числових та текстових: зображення (події, теплові мапи, схема команди тощо), відеозображення (яскраві моменти матчів, інтерв'ю, нагородження тощо). При цьому, сутності предметної області можуть мати безліч зв'язків між собою.

Футбол – командний вид спорту із великою кількістю різних турнірів/змагань локального, національного, міжконтинентального та міжнародного рівня. Кожна команда має велику кількість гравців, деякі з них навіть не грають у матчах, але приписані до клубу. Окрім того футболіст може змінювати клуб раз у пів року та виступати одночасно за дві команди: клуб та національна команда країни.

Кожен гравець може відіграти близько 60 матчів за рік у 5-6 турнірах або змаганнях. Враховуючи, що професіональних гравців у світі зареєстровано більше ніж 250 мільйонів [15], то за рік проходить велика кількість матчів, що є однією із проблем документації у даній області.

Хоча футбол – командний вид спорту, індивідуальна статистика кожного гравця є дуже важливою як для вболівальників, так і для тренерського штабу. Щоб зібрати потрібну статистику для окремого гравця – потрібно обробити велику кількість даних. Також із розвитком технологій під час футбольних матчів збирається все більше і більше даних, від теплових мап до ймовірностей тих чи інших подій.

2.3 Моделювання предметної області

Під час моделювання предметної області було прийняте рішення спроектувати БД, де центральними сутностями є «Гравець» та «Матч» (див. рис. 2.1), оскільки така схема матиме як складні (Гравець-Виступ/ІгровийЕепізод-Матч, Гравець-Клуб-СтатистикаГри-Матч) так і прості зв’язки (Ціна-Гравець). Також на схемі присутні сутності із великою та малою кількостями полів.

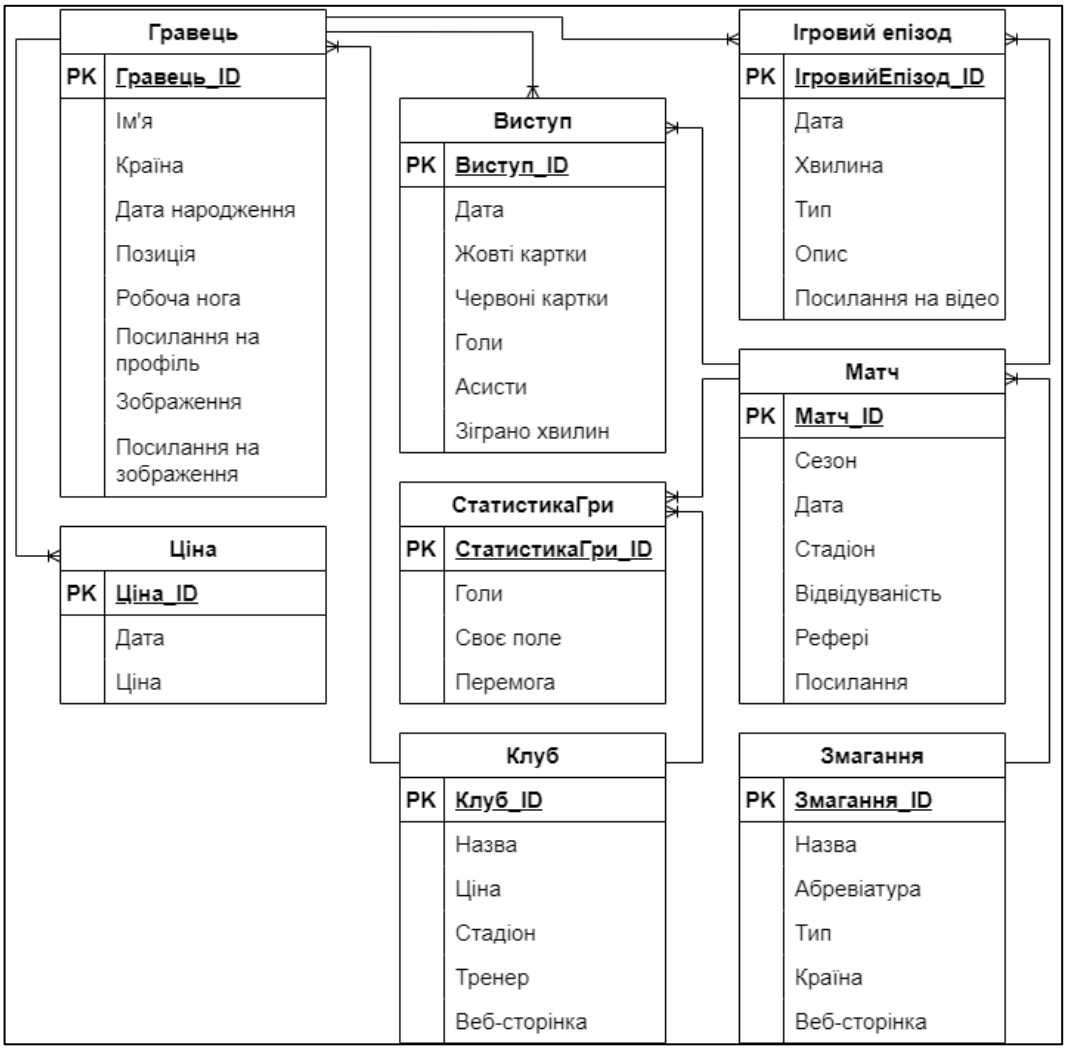

Рисунок 2.1 – ER-діаграма бази даних, що проектується (створено самостійно)

ER-діаграма бази даних, що проектується, містить 5 сутностей: «Гравець» (ім’я, дата народження, країна, робоча нога, зріст, вага, позиція), «Ціна гравця» (дата, ціна), «Матч» (дата, рахунок, відео, стадіон, суддя), «Клуб» (назва, стадіон, тренер), «Змагання» (назва, країна). Кожна сутність має первинний ключ «назва_таблиці_id».

З ER-діаграми видно, що:

- кожен гравець має історію його цін;
- гравець приймає участь у багатьох матчах, у матчі приймає участь багато гравців;
- гравець може змінювати клуб протягом кар’єри, у клубі зареєстровано багато гравців;
- у матчі приймають участь 2 клуби, клуб приймає участь у багатьох матчах;
- клуб може приймати участь у декількох змаганнях, у змаганні приймають участь багато клубів;
- змагання містить багато матчів.

2.3.1 Розробка графової логічної моделі

Основні відмінності графової моделі даних від реляційної [16]:

- мітка вузла – кожна таблиця сутностей у реляційній моделі стає міткою на вузлах у графовій моделі;
- вузол – кожен рядок у реляційній таблиці сутностей стає вузлом у графовій моделі;
- властивість вузла – стовпці (поля) реляційних таблиць стають властивостями вузлів у графовій моделі;
- тільки бізнес-первинні ключі – видалення технічних первинних ключі, залишити бізнес-первинні ключі;
- обмеження/індекси – додавання унікальних обмежень для бізнес-первинних ключів, додавання індексів для атрибутів частого пошуку;
- зв'язки – заміна зовнішніх ключів до іншої таблиці на зв'язки;
- немає значень за замовчуванням – видалити дані зі значеннями за замовчуванням, щоб не зберігати їх;
- очистити дані - дублікати даних у денормалізованих таблицях, можливо, доведеться витягнути в окремі вузли, щоб отримати чистішу модель;

– індексування стовпців в масив - індексовані назви стовпців (наприклад, email1, email2, email3) можуть вказувати на властивості масиву;
– зв'язки - об'єднані таблиці перетворюються на зв'язки, а стовпці цих таблиць стають властивостями зв'язків.

На основі ER-діаграми бази даних (див. рис. 2.1) була спроектована графова модель (див. рис. 2.2). Діаграма схеми графічної БД складається із п'яти вузлів/вершин з атрибутами та 6 ребер/зв'язків, що описують відношення між цими вузлами. Також зв’язки мають назву, на відміну від реляційного підходу. Тому при виконання запитів можна використовувати лише потрібні зв’язки, не обробляючи усю таблицю із потрібними даним при цьому.

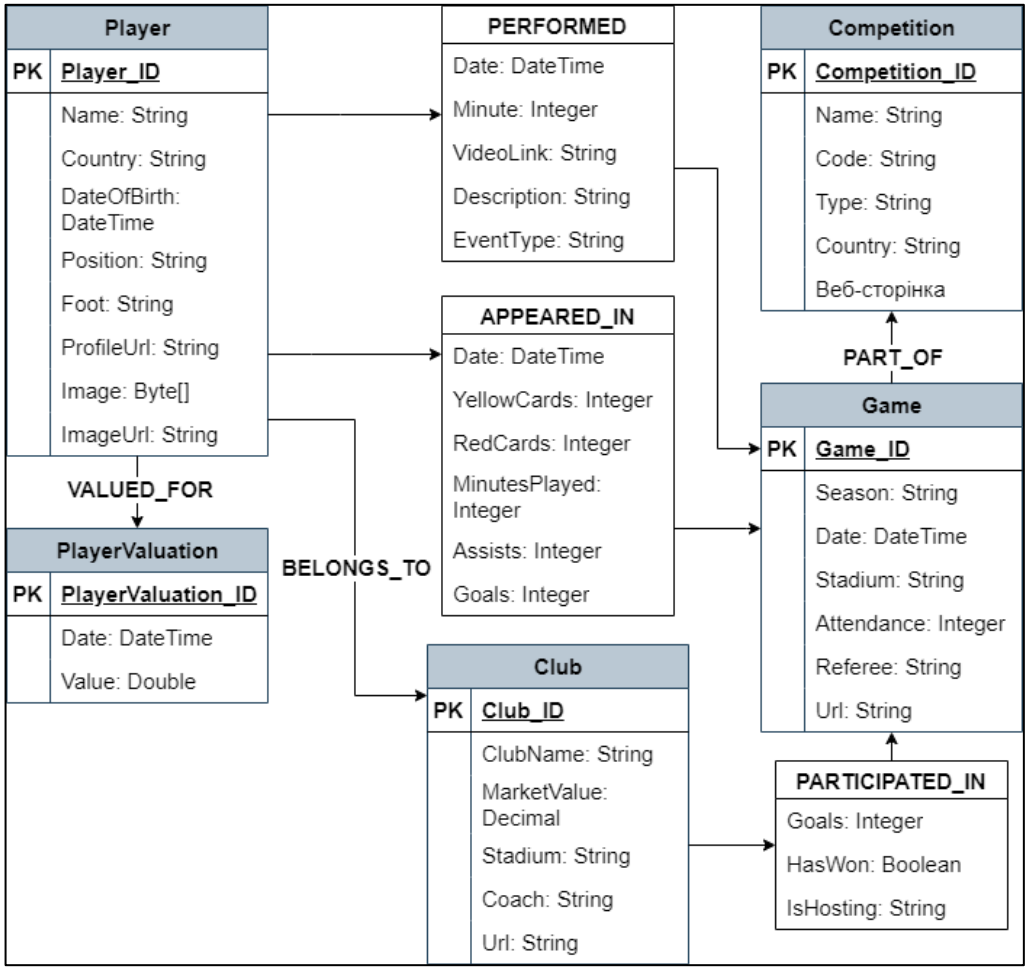

Рисунок 2.2 – Логічна модель графової БД (створено самостійно)

2.3.2 Розробка схеми БД для реляційної моделі

Для моделювання схеми реляційної БД за основу були взяті зв’язки один-до-багатьох та багато-до-багатьох із ER-діаграми бази даних на рисунку 2.1. На

розробленій схемі (див. рис. 2.3) 5 основних таблиць («Гравці» (Players), «Ціни гравця» (PlayerValuations), «Матчі» (Games), «Клуби» (Clubs), «Змагання» (Competitions)) відповідають даній діаграмі, також було додано 3 проміжні таблиці («Appearances», «GameEvents», «ClubGames») для зв'язків багато-до-багатьох у вигляді двох симетричних зв'язків один-до-багатьох. Розроблена схема реляційної БД відповідає третій нормальній формі (3НФ), оскільки жоден атрибут таблиць не залежить інших.

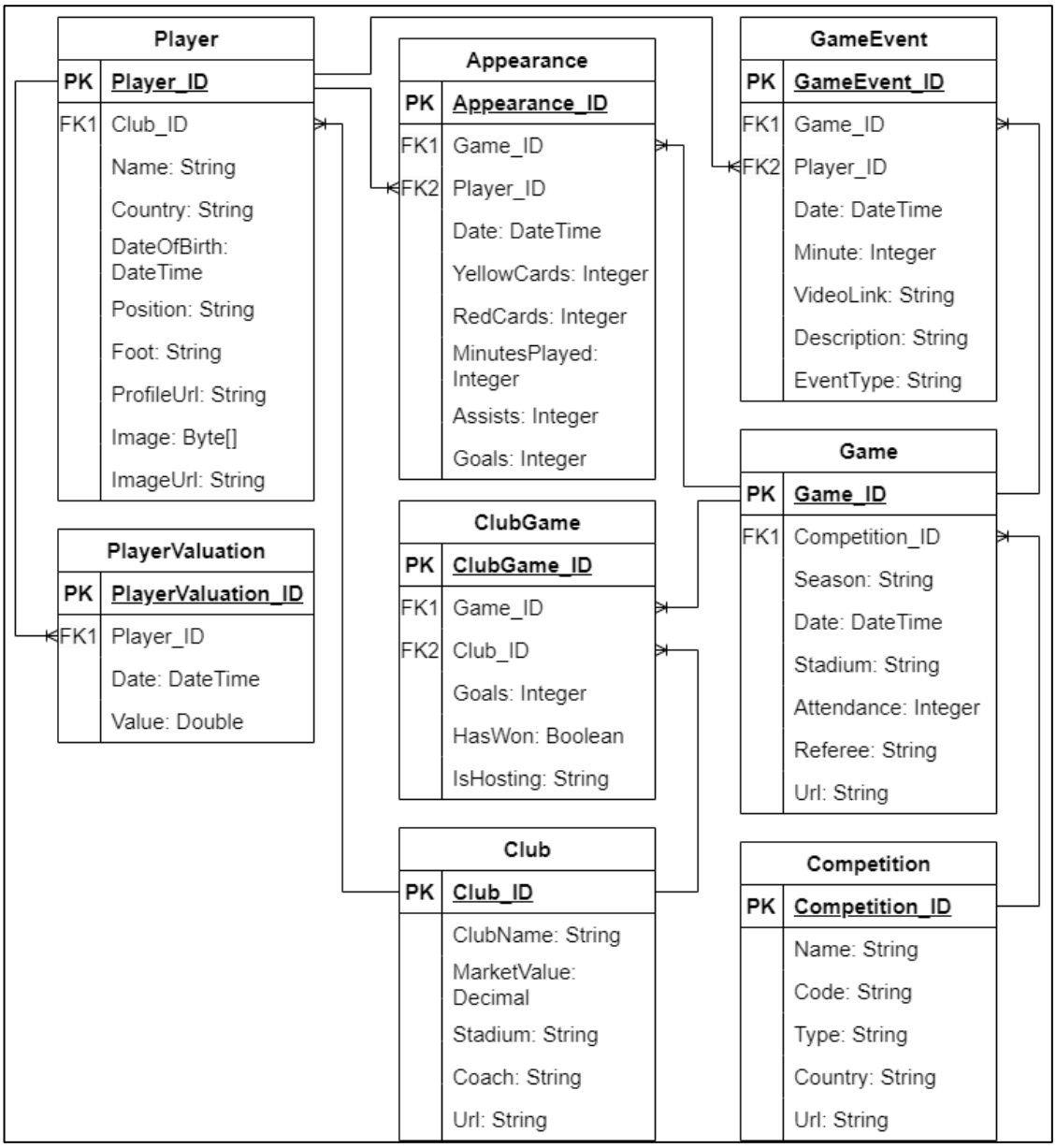

Рисунок 2.3 – Схема реляційної БД (створено самостійно)

З рисунку можна побачити, що графова модель (див. рис. 2.2) простіша за реляційну (див. рис. 2.3) та легка для сприйняття.

2.4 Планування експериментального дослідження

Для проведення дослідження був використаний датасет веб-ресурсу «Kaggle», що надає різноманітні набори даних на будь-яку тематику: https://www.kaggle.com/datasets/davidcariboo/player-scores/versions/292/data.

Датасет містить більш ніж 60 тисяч матчів, більш ніж 400 клубів різного рівня, більш ніж 30 тисяч гравців, більш ніж 400 тисяч записів про ціну гравців у конкретний проміжок часу та більш ніж 1,2 мільйони записів про участь гравців у матчах. Така кількість даних задовольняє умови дослідження. Також датасет містить як зв’язні та і не зв’язні дані. Окрім того, обраний набір даних містить багато інформації різних типів, що збільшує кількість можливих запитів для проведення експерименту.

Для проведення дослідження були обрані CRUD запити, оскільки вони є більш трудомісткими та використовують більше ресурсів за SELECT-запити.

Під час виконання запитів будуть вимірюватись метрики: час виконання запиту (мс), відсоток використання процесора (%), використання оперативної пам’яті (МБ).

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

3.1 Розробка фізичної моделі реляційної БД

Для роботи була обрана СКБД Microsoft SQL Server 15.0.4153 та мова програмування C# платформи .NET з технологією доступу до даних Entity Framework. Технологія Entity Framework замінює класичні SQL-команди на зручний інтерфейс, що дуже просто інтегрується у проект.

Для створення БД використовувався підхід «Code First» [17], тобто спочатку були створені сутності із відповідними атрибутами та зв’язками, і вже на їх основі технологія Entity Framework створила БД для СКБД Microsoft SQL Server.

Приклад сутності «Змагання» (Competition) наведено нижче:

```
public class Competition : IEntity
{
    [Key]
    public string Id { get; set; }
    public string Name { get; set; }
    public string Type { get; set; }
    public string Country { get; set; }
    public string Confederation { get; set; }
    public string Url { get; set; }

    public List<Club> Clubs { get; set; }
    public List<Game> Games { get; set; }
}
```

Для таких моделей створюється таблиця у БД та встановлюються зв’язки із іншими.

Після того як БД була створена – було проведено імпорт даних із обраного датасету. Приклад такого методу для файлу із сутностями «Змагання» (Competition) наведено нижче:

```
try
{
    using (var reader = new
StreamReader("C:\\Users\\Flash\\Desktop\\Transf\\competitions.csv"))
    using (var csv = new CsvReader(reader,
CultureInfo.InvariantCulture))
    {
        var records = csv.GetRecords<Competition>().ToList();
```

```
            _context.Set<Competition>().AddRange(records);
            _context.SaveChanges();
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine($"An error occurred: {ex.Message}");
        return BadRequest(ex.Message);
    }
    return Ok("All done!");
```

Даний метод зчитує рядки .csv файлу та надає значення відповідним атрибутам сутності за полями.

Створена БД наведена на рисунку 3.1.

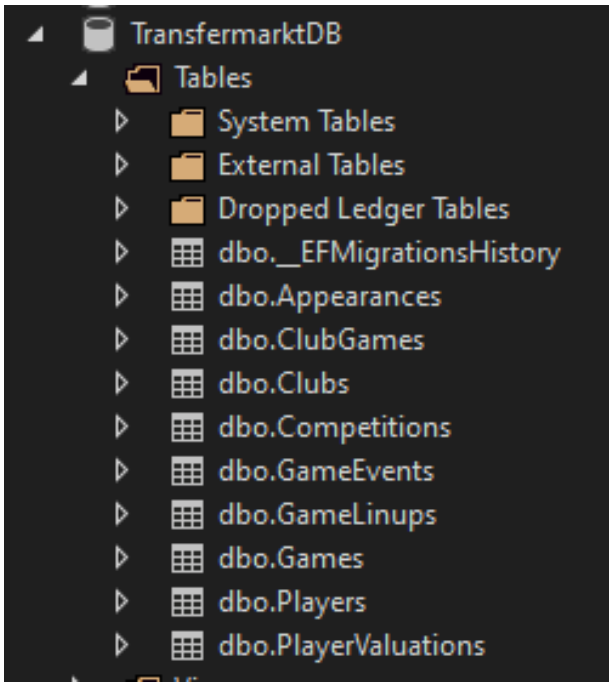

Рисунок 3.1 – Створена реляційна БД (створено самостійно)

З рисунку видно, що було створено 9 таблиць, 5 із яких основні та представлені на рисунку 2.1 на ER-діаграмі бази даних, що проектується; а інші 4 були описані у пункті 3.3.2 та зображені на рисунку 3.2. Оновлення структури БД відбувається за допомогою міграцій. Файл БД має розмір 1,44 ГБ та лог-файл розміром 1,38 ГБ.

Для роботи із базою даних було створено .NET-проект із використанням фреймворку Entity Framework Core.

3.2 Розробка фізичної моделі графової БД

Графова БД була створена у середовищі СКБД Neo4j 5.12.0 та була інтегрована у програмну систему на платформі .NET за допомогою драйвера Дані було імпортовано із csv-файлів раніше зазначеного датасету. Приклад створення вершин що відповідають таблиці Клуб ER-діаграми:

```
LOAD CSV WITH HEADERS FROM 'file:///Players.csv' AS row
CREATE (:Club {
    ClubId: toInteger(row.ClubId),
    Name: row.Name,
    TotalMarketValue: toFloat(row.TotalMarketValue),
    Stadium: row.Stadium,
    Coach: row.Coach,
    Url: row.Url
})
```

Створення вершин таблиці «Ціна Гравця» із ребрами (зв’язком) «VALUED_FOR» до вершин таблиці «Гравець»:

```
LOAD CSV WITH HEADERS FROM 'file:///Valuations.csv' AS row
WITH row, toInteger(row.PlayerId) AS playerId
MATCH (player:Player {PlayerId: playerId})
CREATE (pv:PlayerValuation {
    PlayerValuationId: row.PlayerValuationId,
    Date: row.Date,
    Value: toInteger(row.Value)
})
CREATE (pv)-[:VALUED_FOR]->(player)
```

Створення ребер «PARTICIPATED_IN» між вершинами «Матч» та «Клуб» із властивостями:

```
LOAD CSV WITH HEADERS FROM 'file:///ClubGames.csv' AS row
MATCH (game:Game { GameId: toInteger(row.GameId) })
MATCH (club:Club { ClubId: toInteger(row.ClubId) })
CREATE (club)-[cg:PARTICIPATED_IN {
    ClubGameId: row.ClubGameId,
    Goals: toInteger(row.Goals),
    IsHosting: row.IsHosting,
    HasWon: toBoolean(row.HasWon)
}]->(game)
```

Візуально зв’язок «APPEARED_IN» наведено на рисунку 3.2.

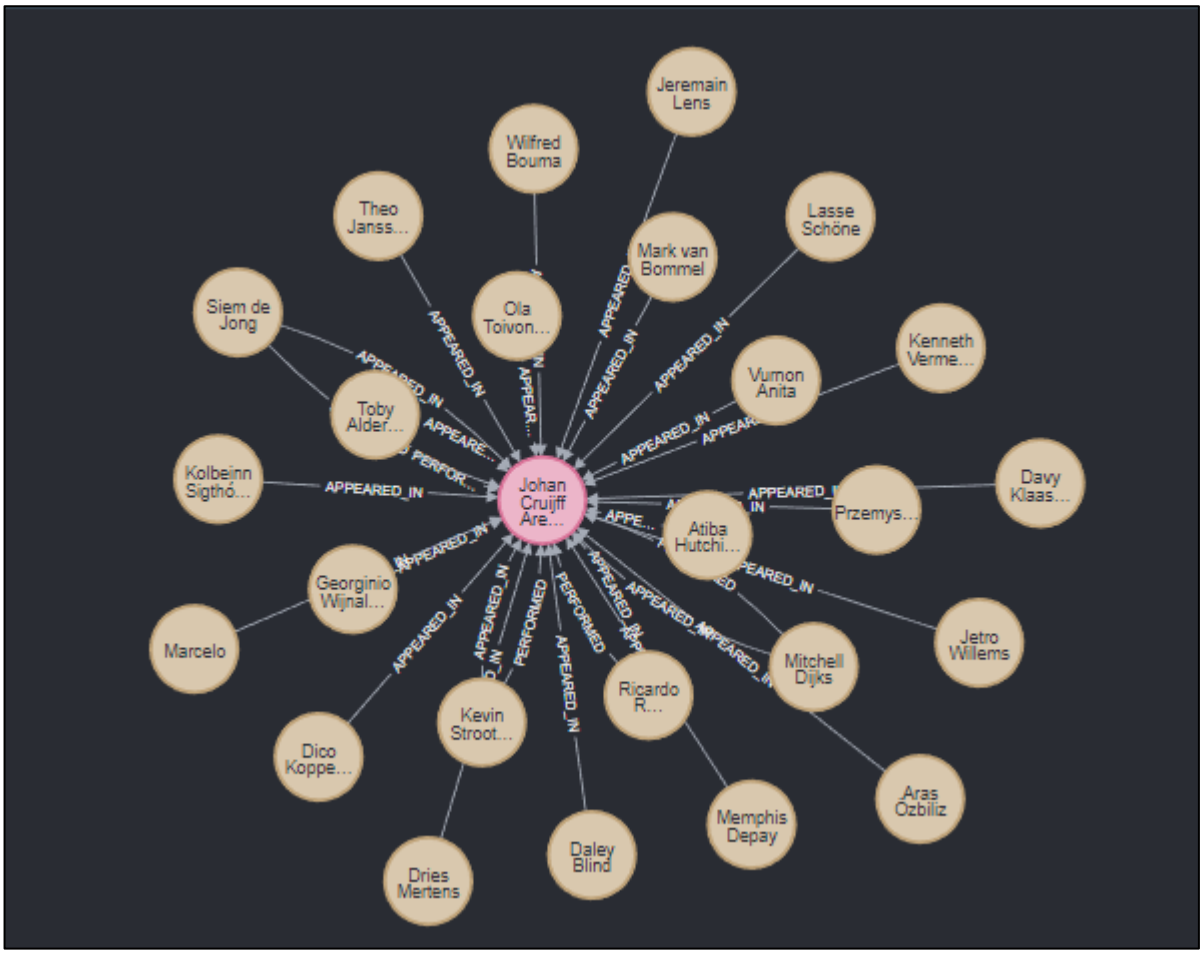

Рисунок 3.2 – Візуальне представлення зв’язку «APPEARED_IN» у графовій БД (створено самостійно)

Розмір БД – 2,42 ГБ, де 974 МБ даних та 1,04 ГБ транзакцій.

Для роботи із базою даних було створено .NET-проект із використанням бібліотеки Neo4j.Driver. З’єднання із сервером СКБД Neo4j забезпечує клас Neo4jDataAccess:

```
public class Neo4jDataAccess : INeo4jDataAccess
{
    private IAsyncSession _session;

    private ILogger<Neo4jDataAccess> _logger;

    private string _database;

    public Neo4jDataAccess(IDriver driver, ILogger<Neo4jDataAccess>
logger, IOptions<ApplicationSettings> appSettingsOptions)
    {
        _logger = logger;
```

```
            _database = appSettingsOptions.Value.Neo4jDatabase ??
"neo4j";
            _session = driver.AsyncSession(o =>
o.WithDatabase(_database));
        }
    }
```

Даний клас містить кілька універсальних методів-конструкторів різних типів універсальних транзакцій для їх виконання на сервері БД. Приклад одного із таких методів, а саме методу виконання запиту до СКБД Neo4j із передачею відповідних параметрів наведено нижче:

```
    public async Task ExecuteWriteTransactionAsync(string query,
IDictionary<string, object>? parameters = null)
    {
        try
        {
            parameters = parameters ?? new Dictionary<string, object>();

            await _session.WriteTransactionAsync(async tx =>
            {
                await tx.RunAsync(query, parameters);
            });
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "There was a problem while executing
database query");
            throw;
        }
    }
```

3.3 Реалізація запитів для експериментального дослідження

Для вимірювання метрик виконуваних запитів було створено бібліотеку на мові програмування C#. Для кожної СУБД також було створено окремий проект на платформі .NET, що реалізовує CRUD-функціоональність. Бібліотека містить 2 методи, що викликаються відповідно до, та після виконання запиту та надають інформацію вимірюввані метрики. Відслідковування метрик «використання оперативної пам’яті» та «використання процесора» реалізована через клас System.Process:

```
    public QueryPerformanceMonitor()
```

```
    {
        process = Process.GetCurrentProcess();
    }

    public void StartMonitoring()
    {
        initialCpuTime = process.TotalProcessorTime;
        initialMemory = process.WorkingSet64;
        stopwatch = Stopwatch.StartNew();
    }

    public QueryPerformanceMetrics StopMonitoring()
    {
        stopwatch.Stop();
        process = Process.GetCurrentProcess();
        // Get final memory usage of the SQL Server process
        TimeSpan finalCpuTime = process.TotalProcessorTime;

        TimeSpan cpuTimeUsed = finalCpuTime - initialCpuTime;

        TimeSpan totalTime = DateTime.Now - process.StartTime;

        double cpuUsagePercentage = (cpuTimeUsed.TotalMilliseconds /
totalTime.TotalMilliseconds) * 100;

        long finalMemory = process.WorkingSet64;

        // Calculate memory usage difference
        long memoryUsed = (finalMemory - initialMemory) / (1024 * 1024);

        Console.WriteLine($"Memory used by SQL Server during query
execution: {memoryUsed} MB");
        Console.WriteLine($"CPU SQL Server during query execution:
{cpuUsagePercentage}%\nExecution time: {stopwatch.Elapsed.TotalSeconds}");

        return new QueryPerformanceMetrics
        {
            ExecutionTimeMilliseconds =
stopwatch.Elapsed.TotalMilliseconds,
            CpuTimeUsedMilliseconds = cpuTimeUsed.TotalSeconds,
            CpuUsagePercentage = cpuUsagePercentage,
            MemoryUsedMB = memoryUsed
        };
    }
```

Метод StartMonitoring() вимірює метрики перед запитом, а метод StopMonitoring() вимірює їх після виконання запиту та повертає у вигляді об’єкта QueryPerformanceMetrics.

Запити до СКБД Neo4j розроблені на декларативній мові запитів Cypher. Передача запитів до СКБД відбувається за допомогою класу Neo4jDataAccess, що було раніше у цьому розділі.

Запит INSERT використовує оператор $UNWIND, що отримує колекцію елементів як параметр та виконує подальшу частину запиту для кожного із них. Такий підхід було обрано з метою оптимізації продуктивності Neo4j, оскільки у даному випадку до СКБД виконується тільки один запит, а при виконанні окремого запиту для кожного елементу колекції застосунок виконує додаткові запити для забезпечення з'єднання із сервером, на якому розгорнуто БД. Програмний код даного запиту наведено нижче:

```
     using (QueryPerformanceMonitor queryPerformanceMonitor = new
QueryPerformanceMonitor())
     {
         try
         {
             var query = @"
                 UNWIND $players AS player
                 CREATE (p:Player {
                     PlayerId: player.PlayerId,
                     Name: player.Name,
                     Country: player.Country,
                     DateOfBirth: player.DateOfBirth,
                     Position: player.Position,
                     Foot: player.Foot,
                     ImageUrl: player.ImageUrl,
                     //Image: player.Image,
                     Url: player.Url
                 })
                 WITH player, p
                 MATCH (c:Club {ClubId: player.CurrentClubId}), (g:Game
{GameId: player.GameId})
                 CREATE (p)-[:BELONGS_TO]->(c), (p)-[:APPEARED_IN]->(g),
(p)-[:PERFORMED]->(g)
                 ";

             var parameters = new Dictionary<string, object>
                 {
                     { "players", players.Select(player => new
Dictionary<string, object>
                         {
                             { "PlayerId", player.PlayerId },
                             { "Name", player.Name },
                             { "Country", player.Country },
                             { "CurrentClubId", player.CurrentClubId },
                             { "DateOfBirth", player.DateOfBirth },
                             { "Position", player.Position },
                             { "Foot", player.Foot },
                             { "ImageUrl", player.ImageUrl },
                             { "Url", player.Url },
                             { "GameId", 3079396}
                         }).ToList()
                     }
```

```
            };
            queryPerformanceMonitor.StartMonitoring();
            await _neo4jDataAccess.ExecuteWriteTransactionAsync(query,
parameters);
            var metrics = queryPerformanceMonitor.StopMonitoring();
            return metrics;
```

У наведеному фрагменті коду текст запиту представлено у змінній «query».

Для процедури UPDATE був застосований такий же підхід, текст запиту наведено нижче:

```
    UNWIND $players AS player
     MATCH (p:Player { PlayerId: player.PlayerId }), (c1:Club {ClubId:
p.CurrentId}), (g1:Game {GameId: p.GameId}),
          (c2:Club {ClubId: player.CurrentId}), (g2:Game {GameId:
player.GameId}),
          (p) - [rel:BELONGS_TO] - (c1), (p) - [rel2:APPEARED_IN] - (g1),
(p) - [rel3:PERFORMED] - (g1)

     SET p.Name = player.Name,
         p.Country = player.Country,
         p.DateOfBirth = player.DateOfBirth,
         p.Position = player.Position,
         p.Foot = player.Foot,
         p.ImageUrl = player.ImageUrl,
         //p.Image = player.Image,
         p.Url = player.Url
     WITH p, player, g2, c2, c1, g1, rel, rel2, rel3
     CREATE (p)-[:BELONGS_TO]->(c2), (p)-[:APPEARED_IN]->(g2), (p)-
[:PERFORMED]->(g2)
     DETACH DELETE rel, rel2, rel3
```

Для процедури DELETE було розроблено єдиний запит, оскільки єдиним параметром такого запиту є умова видалення, у даному випадку це Id. Умова «WHERE p.PlayerId >= 1200000» використовується через те, що згенеровані тестові дані мають Id більше 1200000, тоді як уже наявні у БД дані мають Id менше за 1200000. Текст запиту:

```
MATCH (p:Player)
WHERE p.PlayerId >= 1200000
DETACH DELETE p
```

SELECT запит для варіації «без зв’язків та без зображень» наведено нижче:

```
MATCH (p:Player)
WHERE p.Name CONTAINS 'layer'
RETURN p
{
    PlayerId: p.PlayerId,
    Name: p.Name,
    CurrentClubId: p.CurrentClubId,
    Country: p.Country,
    DateOfBirth: p.DateOfBirth,
    Position: p.Position,
    Foot: p.Foot,
    ImageUrl: p.ImageUrl,
    Url: p.Url
}
```

Запити до СКБД MSSQL розроблені із використанням фреймфорку EntityFramework Core, що може працювати із колекціями даних, що вирішує проблему використання послідовних запитів для кожної окремої сутності. Запит INSERT для всієї колекції даних у коді виглядає наступним чином:

```
monitor.StartMonitoring();
context.Database.ExecuteSqlRaw("SET IDENTITY_INSERT dbo.Players
ON;");
await _context.Set<Player>().AddRangeAsync(players);
await _context.SaveChangesAsync();
_context.Database.ExecuteSqlRaw("SET IDENTITY_INSERT dbo.Players
OFF;");
transaction.Commit();
var metrics = monitor.StopMonitoring();
return Ok($"All done! \nMemory used by SQL Server during query
execution: {metrics.MemoryUsedMB} MB\n" +
    $"CPU SQL Server during query execution:
{metrics.CpuUsagePercentage}%\n" +
    $"Execution time: {metrics.ExecutionTimeMilliseconds} ms");
```

Метод AddRangeAsync() фреймфорку EntityFramework Core, розбиваючи колекцію даних на менші частини, паралельно виконує декілька запитів із параметрами до СКБД. Приклад такого запиту наведено нижче:

```
INSERT INTO [Players] ([PlayerId], [Country], [CurrentClubId],
[DateOfBirth], [Foot], [Image], [ImageUrl], [Name], [Position], [Url])
      VALUES (@p0, @p1, @p2, @p3, @p4, @p5, @p6, @p7, @p8, @p9),
```

Для розробки запиту UPDATE було застосовано такий же підхід:

```
        monitor.StartMonitoring();
        context.Set<Player>().UpdateRange(players);
        await _context.SaveChangesAsync();
        transaction.Commit();
        var metrics = monitor.StopMonitoring();
        return Ok($"All done! \nMemory used by SQL Server during query
execution: {metrics.MemoryUsedMB} MB\n" +
            $"CPU SQL Server during query execution:
{metrics.CpuUsagePercentage}%\n" +
            $"Execution time: {metrics.ExecutionTimeMilliseconds} sec");
```

Приклад прямого запиту до СКБД:

```
        UPDATE [Players] SET [Country] = @p310, [CurrentClubId] = @p311,
[DateOfBirth] = @p312, [Foot] = @p313, [Image] = @p314, [ImageUrl] = @p315,
[Name] = @p316, [Position] = @p317, [Url] = @p318
              WHERE [PlayerId] = @p319;
              SELECT @@ROWCOUNT;
```

Для процедури DELETE було розроблено прямий запит до СКБД:

```
        _context.Database.ExecuteSqlRaw("DELETE FROM[Players] WHERE[PlayerId]
>= 1200000;");
        _context.SaveChanges();
```

Як і у розробленому для Neo4j запиті, умова «WHERE[PlayerId] >= 1200000» використовується через те, що згенеровані тестові дані мають Id більше 1200000, тоді як уже наявні у БД дані мають Id менше за 1200000.

SELECT запит для варіації «без зв’язків та без зображень» наведено нижче:

```
        string NoRelsNoImagesQuery = @"
            SELECT PlayerId, CurrentClubId, Name, Country, DateOfBirth,
Position, Foot, ImageUrl, Url
            FROM [Players]
            WHERE [Name] LIKE '%layer%'
        ";
```

Усього було розроблено 8 запитів (по 4 для кожної СКБД). Також для кожного запиту було розроблено по 4 його варіації, що будуть детально описані у наступному розділі.

4 ОПИС ЕКСПЕРЕМЕНТАЛЬНИХ ДОСЛІДЖЕНЬ

4.1 Проведення експериментальних досліджень

Відповідно до запропонованого плану було проведено дослідження продуктивності роботи реляційної СКБД MSSQL та графової СКБД Neo4j.

Для кожного із 4-ох типів запитів було розроблено 4 варіації, де кожен запит використовує сутності, що:

- мають по 4 активних зв’язки та використовують зображення;
- мають по 4 активних зв’язки та не використовують зображення;
- не мають активних зв’язків та використовують зображе;
- не мають активних зв’язків та не використовують зображення.

Усього досліджено по 48 запитів (4 варіації, 4 типи та 3 розміри вибірки) до кожної СКБД. Розміри вибірок наступні: 10000, 25000, 50000.

Під час виконання запитів вимірювались метрики: час виконання запиту (ms), відсоток завантаження процесора (%), використання оперативної пам’яті (МБ).

Результати дослідження показали, що Neo4j втрачає продуктивність при великій кількості послідовних запитів: тестовий запит із використанням оператора UNWIND, що обробляє всю колекцію сутностей за один запит, виконався за 4 секунди для 10000 елементів. У той час як 10000 послідовних запитів для кожного окремого елементу виконувались 57 секунд. Також було виявлене значне зростання часу виконання при обробці сутностей із GUID (статистично унікальний 128-бітний ідентифікатор).

Під час проведення першого експерименту досліджувались INSERT запити для MSSQL та CREATE запити для Neo4j (оскільки у Neo4j не потрібно завчасно створювати схеми даних: у Neo4j CREATE запит створює вузол або ребро заданої структури одразу із уведеними даними). Отримані результати дослідження CREATE-запитів наведено у табл. 4.1.

Таблиця 4.1 – Результати виконання запитів створення сутностей гравця (створено самостійно)

СКБД та запит /Метрики	Швидкість виконання запиту (мс)			Відсоток використання процесора (%)			Витрати оперативної пам’яті (МБ)		
	10000	25000	50000	10000	25000	50000	10000	25000	50000
MSSQL (зв’язки + зображення)	18757	47170	100844	18,15	37,41	74,81	261,2	551,5	1081,3
MSSQL (зв’язки, без зображень)	18232	46835	97738	16,53	35,3	74,96	237,6	536,9	1065,5
MSSQL (без зв’язків + зображення)	2808	6259	12670	13,69	22,12	27,56	130,2	442	798,8
MSSQL (без зв’язків, без зображень)	2403	5611	11171	11,94	22,32	32,39	129, 5	444,1	893,8
Neo4j (зв’язки + зображення)	12913	31433	44778	6,63	6,72	6,84	68,8	209,1	606,6
Neo4j (зв’язки, без зображень)	857	1491	2630	5,98	6,65	6,77	31,3	57	153
Neo4j (без зв’язків + зображення)	12672	31384	44484	4	3,28	4,8	69,8	205,6	610
Neo4j (без зв’язків, без зображень)	566	946	1686	4,72	3,16	4,15	29,6	71,2	145,9

Результати проведеного дослідження показали, що на продуктивність MSSQL найбільше впливає зв’язність даних, тоді як наявність зображень (великий за розміром тип даних) – майже не впливає; на продуктивність Neo4j впливає наявність зображень, а зв’язність даних – майже не впливає. Відсоток використання процесора та використання оперативної пам’яті у всіх випадках більші у MSSQL. Якщо витрати RAM можна пояснити тільки особливостями роботи MSSQL та EntityFramework Core, то вищий показник відсотку використання процесора можна пояснити тим, що на відміну від запитів до Neo4j – виконується не один запит – а

декілька запитів із використанням збережених процедур. Наприклад, при виконанні даного запиту для Neo4j без оператора UNWIND відсоток використання процесора знаходиться в межах 70-90%. Час виконання запитів до Neo4j менший, ніж до MSSQL в усіх випадках, крім варіації запиту «без зв'язків + зображення».

Слід зазначити, що при використанні сторонніх бібліотек, наприклад, EFCore.BulkExtensions [18] – можна збільшити продуктивність роботи MSSQL, проте в такому разі значно знижується точність вимірювання досліджуваних метрик розробленим програмним забезпеченням.

Більш наочно результати експерименту представлені на графіку (див. рис. 4.1), що демонструє залежність часу виконання запитів до MSSQL від зв'язності даних та до Neo4j від наявності зображень.

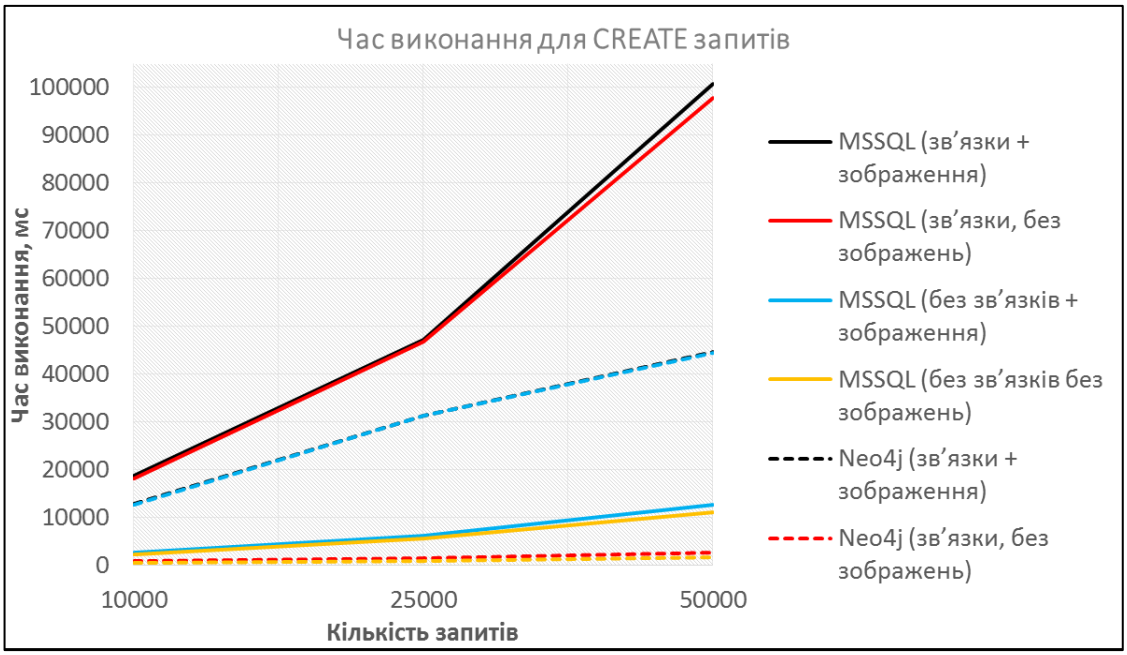

Рисунок 4.1. – Графік із результатами першого експерименту (створено самостійно)

Прямі, що стосуються СКБД MSSQL на графіку зображені суцільними лініями, прямі для Neo4j – пунктиром; також для зручності запити у ідентичних сценаріях для обох СКБД зображені прямими одного кольору. Графік демонструє розподілення прямих на 4 групи: по 2 прямі, що відповідають хорошим та поганим сценаріям для кожної з СКБД.

Результати проведених дослідів для UPDATE-запитів наведено у табл. 4.2. Результати даного дослідженя повторюють тенденції для Neo4j із попереднього досліду: на продуктивність СКБД впливає обсяг даних та наявність зображень, зв'язність даних майже не впливає. Також отримані метрики подібні до метрик із попереднього досліду. У порівнянні із результатами у табл. 1 MSSQL продуктивніша при виконанні UPDATE-запитів; тенденція залежності продуктивності від параметрів досліду змінилась – зв'язність даних та наявність зображень мають майже однаковий вплив на досліджувані метрики.

Таблиця 4.2 – Результати виконання запитів оновлення сутностей гравця (створено самостійно)

СКБД та запит /Метрики	Швидкість виконання запиту (мс)			Відсоток використання процесора (%)			Витрати оперативної пам’яті (МБ)		
	10000	25000	50000	10000	25000	50000	10000	25000	50000
MSSQL (зв’язки + зображення)	4836	8639	21679	19,96	22,29	37,60	282,51	483,88	843,61
MSSQL (зв’язки, без зображень)	3359	8213	18973	12,22	18,93	29,69	251,95	452,10	838,85
MSSQL (без зв’язків + зображення)	3444	8322	16253	13,51	24,48	33,32	192,79	399,84	774,86
MSSQL (без зв’язків без зображень)	3346	7789	15453	14,7	23,77	27,48	158,53	370,7	741,96
Neo4j (зв’язки + зображення)	14638	38311	51910	6,61	6,8	7,1	140,16	392,94	904,61
Neo4j (зв’язки, без зображень)	374	773	1730	6,16	6,37	6,67	29,6	72,22	146,96
Neo4j (без зв’язків + зображення)	16297	37265	49492	6,42	6,54	7,29	145,63	492,66	894,16
Neo4j (без зв’язків без зображень)	368	749	1745	6,11	6,22	6,69	25,38	68,98	145,85

Під час проведення досліду щодо запиту оновлення Neo4j виявилась продуктивнішою за MSSQL лише при виконанні запитів, що не містять зображень. MSSQL для всіх 4 варіацій запитів показала приблизно однаковий результат, із незначною перевагою запитів, де фігурують дані із слабкою зв’язністю.

Під час проведення даного досліду не було виявлено однозначних переваг у використанні оперативної пам’яті або часі виконання запитів жодної із СКБД, проте відсоток використання процесора вдруге виявився меншим у Neo4j.

Результати проведених дослідів для DELETE-запитів наведено у табл. 4.3.

Таблиця 4.3 – Результати виконання запитів видалення сутностей гравця (створено самостійно)

СКБД та запит /Метрики	Швидкість виконання запиту (мс)			Відсоток використання процесора (%)			Витрати оперативної пам’яті (МБ)		
	10000	25000	50000	10000	25000	50000	10000	25000	50000
MSSQL (зв’язки + зображення)	1549	2369	4138	8,84	10,8	8,89	23,59	23,66	23,72
MSSQL (зв’язки, без зображень)	1234	1582	3082	8,7	11,83	11,83	23,47	23,61	23.71
MSSQL (без зв’язків + зображення)	1346	2387	3643	11,76	11,75	11,95	23,44	23,62	23,72
MSSQL (без зв’язків без зображень)	1262	1605	1998	11,44	11,56	11,71	23,42	23,62	23,68
Neo4j (зв’язки + зображення)	1077	3221	9157	6,21	6,68	7,75	2,49	2,61	2,94
Neo4j (зв’язки, без зображень)	781	2297	7558	5,15	5,35	5,68	2,44	2,5	2.64
Neo4j (без зв’язків + зображення)	793	1741	3467	6,01	6,53	7,32	2,3	2.37	2,6
Neo4j (без зв’язків без зображень)	400	692	1218	4,38	5,07	5,35	2,28	2,39	2,43

Під час даного досліду кращий відсоток використання процесора знову показала Neo4j, витрати оперативної пам’яті також менші у Neo4j. Швидкість виконання запитів вища у Neo4j при меншій кількості сутностей та у MSSQL при більшій кількості. Однозначна тенденція впливу зв’язності даних та наявності зображень відсутня, оскільки у різних сценаріях обидва параметри впливають на продуктивність СКБД по-різному. У порівнянні із попередніми експериментами різниця між метриками у даному досліді набагато менша між усіма сценаріями.

Четвертий експеримент виконувався для SELECT запитів (див. табл 4.4).

Таблиця 4.4 – Результати виконання запитів пошуку сутностей гравця (створено самостійно)

СКБД та запит /Метрики	Швидкість виконання запиту (мс)			Відсоток використання процесора (%)			Витрати оперативної пам’яті (МБ)		
	10000	25000	50000	10000	25000	50000	10000	25000	50000
MSSQL (зв’язки + зображення)	3273,6	4874,9	8761,7	20,96	22,23	23,31	173,55	347,64	690,63
MSSQL (зв’язки, без зображень)	3228,7	4938,8	7925	18,85	19,75	22,73	81,33	169,93	231,61
MSSQL (без зв’язків + зображення)	1722,4	2051,7	2561,5	15,25	19,15	21,51	125,48	334,99	613,27
MSSQL (без зв’язків, без зображень)	1586	1844	2047,9	14,84	20,96	21,56	49,12	71,46	82,42
Neo4j (зв’язки + зображення)	651,4	975,4	1889,4	17,86	20,41	28,60	174,84	375,47	697,73
Neo4j (зв’язки, без зображень)	513,7	674,7	1073,3	7,28	9,97	13,64	52,87	144,52	309,16
Neo4j (без зв’язків + зображення)	574,6	790,1	1638,3	3,07	8,16	18,34	159,95	342,18	495,24
Neo4j (без зв’язків, без зображень)	444,1	483,8	923,2	3,96	5,50	15,12	35,36	65,55	182,23

Під час проведення даного досліду кращий відсоток використання процесора знову показала Neo4j, проте Neo4j також зафіксував найвищий показник за даною метрикою у досліді. Витрати оперативної пам’яті менші у Neo4j при роботі без зображень, та менші у MSSQL при роботі з зображеннями. Швидкість виконання запитів вища у Neo4j. Результати досліду SELECT запитів продовжують тенденції попередніх дослідів.

У всіх дослідах Neo4j показала найкращий показник процесорного відсотка та майже у всіх – найкращий показник використання оперативної пам’яті. MSSQL виявилась продуктивнішою за показником «час виконання запиту» майже у всіх дослідах із використанням зображень, крім сценарію «зв'язки + зображення» при дослідженні CREATE-запитів, та за показником «використання оперативної пам'яті» при виконанні UPDATE-запитів при використанні зображень.

У ході дослідження були виявлені наступні залежності між даними та продуктивністю СКБД:

- СКБД MSSQL найкраще підходить для роботи із великими за розміром типами даних, таких як двійкові дані у вигляді масивів байтів, при повній відсутності зв’язків у сутності, та втрачає продуктивність при збільшенні кількості зв’язків до 4-ьох. Також MSSQL для своєї роботи використовує більше ресурсів комп’ютера за Neo4j, у деяких випадках цей показник для MSSQL більший у 10 разів. Як результат, обсяг доступних ресурсів також слід мати на увазі при виборі СКБД для розробки;
- СКБД Neo4j має високу продуктивність при роботі як із сутностями що не мають зв’язків, так і з сутностями, що мають 4 та більше зв’язків. Також на продуктивність роботи Neo4j майже не впливає кількість оброблюваних даних (дослідженні проводилось для вибірок із 10000, 25000 та 50000 елементів), проте має проблеми при роботі із типами даних, таких як великі масиви байтів та GUID;
- СКБД Neo4j має проблеми із продуктивністю при виконанні від 10000 та більше послідовних запитів.

На основі проведеного аналізу сформовані рекомендації щодо вибору СКБД в залежності від наявних даних:

- СКБД MSSQL найкраще підходить для роботи із структурованими даними, що мають менше 4 зв’язків та добре себе показує при обробці двійкових даних. MSSQL витрачає близько 1 ГБ та навантажує процесор на 30-40% для представлення даних у вигляді таблиць та зв’язків між ними, проте використання великих за розміром (до 100 КБ) двійкових даних не впливає ні на час виконання запитів, ні на виділення ресурсів комп’ютера. Проблема високого використання ресурсів комп’ютера даною СКБД на фоні конкурентів частково вирішується при її роботі із зображеннями, аудіо та відеофайлами, оскільки на відміну від MSSQL інші СКБД потребують додаткових ресурсів для роботи із ними;
- СКБД Neo4j добре показала себе при роботі як із даними без зв’язків, так і з даними, що мали 4 зв’язки.. Проте при використанні зображень різниця із цими ж запитами без зображень у швидкості виконання запитів становить близько 15 разів, тоді як витрати оперативної пам’яті збільшуються у близько 5 разів. Підсумовуючи усе вищесказане, дана альтернатива підходить як для систем із простою ієрархією даних, так і для систем із складними зв’язками, де сутність може мати більше 4-ьох зв’язків, проте використання Neo4j для вибірок із двійковими даними доцільне лише при їх високій зв’язності (від 4 та більше зв’язків на сутність).

Розроблені рекомендації можуть бути використані під час розробки програмних систем на платформі .NET зі структурою даних, подібною до тієї, що розглядалась у дослідженні. Також дослідження приділяє увагу в основному зображенням та майже не розглядає взаємодію СКБД із іншими типами, оскільки масиві байтів (представлення зображення у БД) мають значно більший розмір, а тому сильніше впливають на продуктивність роботи СКБД. Подальші дослідження можуть поглибити дослідження щодо інших типів даних, а також дослідити результати при іншому підході до зберігання двійкових даних.

ВИСНОВКИ

У ході дослідження проведено аналіз предметної галузі, розроблено постановку задачі та вирішено проблеми, що пов’язані з прийняттям проектних рішень.

У роботі було проведено аналіз існуючих СКБД та вирішена багатокритеріальна задача з вибору підходящої СКБД для подальшого дослідження. Було проведено аналіз графових СКБД та серед них була обрана найоптимальніша (Neo4j).

Було проведено планування експериментальної частини дослідження, а саме: обрана предметна галузь дослідження, проаналізовано умови проведення дослідження. Для проведення дослідження була обрана предметна область футбольної статистики, оскільки вона задовольняє умови його проведення. Було проведено аналіз предметної галузі та була описана програмна реалізація реляційної БД.

Було розроблено ER-діаграми графової та реляційної баз даних, на основі яких та були створені їх фізичні моделі. Обидві БД було заповнено даними із об’ємного датасету, а потім інтегровано до .NET проектів для проведення подальших досліджень. Було розроблено бібліотеку для вимірювання потрібних для дослідження метрик (відсоток використання процесора, використання оперативної пам’яті та загальний час виконання) під час виконання запитів. Було розроблено по 3 запити та 4 їх варіації до кожної із досліджуваних СКБД.

На основі результів дослідженні були сформовані рекомендації щодо сценаріїв використання реляційної СКБД MSSQL та графової Neo4j. У ході експериментів було виявлено, що у контексті дослідження на продуктивність MSSQL впливає кількість зв’язків, тоді як на продуктивність Neo4j впливають типи даних, що використовуються.

За результатами дослідження опубліковано тези «Investigation of the Efficiency Dependence of Relational and Graph Databases on Data» на 12-тій Міжнародній науковій та технічній конференції «Інформаційні системи та технології» (IST-2023) (див. додаток Г).

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. What is a Relational Database (RDBMS)? URL: https://www.oracle.com/database/what-is-a-relational-database/#:~:text=A%20relational%20database%20is%20a,of%20representing%20data%20in%20tables (дата звернення: 20.04.2024).

2. SQL Datatype. URL: https://www.javatpoint.com/dbms-sql-datatype (дата звернення: 25.04.2024).

3. Introduction to NoSQL. URL: https://www.geeksforgeeks.org/introduction-to-nosql/ (дата звернення: 25.04.2024).

4. Mazurova, O. Research of ACID transaction implementation methods for distributed databases using replication technology / Mazurova, O., Naboka, A., Shirokopetleva, M. Innovative technologies and scientific solutions for industries, (2 (16), pp. 19-31. Doi: 10.30837/ITSSI.2021.16.019.

5. Mazurova, O. NOSQL database logic design methods for MONGODB and NEO4J / Mazurova, O., Syvolovskyi, I., Syvolovska, O. Innovativ technologies and scientific solutions for industries, № 2 (20), pp. 52-63.

6. Graph Database Defined. URL: https://www.oracle.com/eg/autonomous-database/what-is-graph-database/ (дата звернення: 23.04.2024).

7. Poulton N. Data Storage Networking: Real World Skills for the Comptia Storage+ Certification and Beyond. Wiley & Sons, Incorporated, John, 2014.

8. Kotiranta P., Junkkari M., Nummenmaa J. Performance of graph and relational databases in complex queries. Applied sciences. 2022. T. 12, № 13. C. 6490. URL: https://doi.org/10.3390/app12136490 (дата звернення: 03.05.2024).

9. Lazarska M., Siedlecka-Lamch O. Comparative study of relational and graph databases. 2019 IEEE 15th international scientific conference on informatics, м. Poprad, Slovakia, 20–22 листоп. 2019 р. 2019. URL: https://doi.org/10.1109/informatics47936.2019.9119303 (дата звернення: 07.05.2024).

10. Stanescu, Liana. (2021). A Comparison between a Relational and a Graph Database in the Context of a Recommendation System. 133-139. URL: https://doi.org/10.15439/2021F33.

11. RedisGraph GraphBLAS Enabled Graph Database. URL: https://www.researchgate.net/publication/332873914_RedisGraph_GraphBLAS_Enabled_Graph_Database (дата звернення: 20.04.2024).

12. Lets Compare – Amazon Neptune vs ArangoDB vs Cassandra vs Cosmos DB vs Neo4j. URL: https://statusneo.com/lets-compare-amazon-neptune-vs-arangodb-vs-cassandra-vs-cosmos-db-vs-neo4j/ (дата звернення: 20.05.2024).

13. OrientDB Community. URL: https://orientdb.org/ (дата звернення: 21.05.2024).

14. Azure Cosmos DB Performance: Direct vs. Gateway mode. URL: https://krishsub.medium.com/azure-cosmos-db-performance-direct-vs-gateway-mode-c06e30f97c1c (дата звернення: 22.05.2024).

15. Introduction & Top Questions. URL: https://www.britannica.com/sports/football-soccer (дата звернення: 22.05.2024).

16. Data Model Transformation Tips. URL: https://neo4j.com/developer/relational-to-graph-modeling/ (дата звернення: 20.05.2024).

17. What is Code-First? URL: https://www.entityframeworktutorial.net/code-first/what-is-code-first.aspx (дата звернення: 24.05.2024).

18. EFCore.BulkExtensions URL: https://github.com/borisdj/EFCore.BulkExtensions (дата звернення: 15.05.2024).