МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет радіоелектроніки
Факультет Комп'ютерних наук
Кафедра Програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

другий (магістерський)
(рівень вищої освіти)

Дослідження методів підтримки міграції для різних моделей зберігання даних

Виконала:
Студентка 2 курсу групи ІПЗм-20-2
Перетятко М. В.
(прізвище, ініціали)

Спеціальність 121 — Інженерія програмного забезпечення

Тип програми Освітньо-наукова

Керівник проф. Лєсна Н.С.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. Кафедри _____ З. В. Дудар

2022

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

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
« ____ » _____ 2022 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентки Перетятко Марії Вікторівни
(прізвище, ім'я, по батькові)

1. Тема роботи: «Дослідження методів підтримки міграції для різних моделей зберігання даних»
затверджена наказом університету від «24» 03 2022 р. №412Ст
2. Термін подання студентом роботи до екзаменаційної комісії «10» травня 2022 р
3. Вихідні дані до роботи *методичні вказівки до виконання кваліфікаційної роботи, літературні джерела по темі міграції баз даних, вимоги до змістового наповнення звіту, вимоги щодо наявності програмної реалізації, що автоматизує процес міграції схеми даних.*
Використовувати ОС Windows 10, СКБД MS SQL Server 2019, MongoDB 4.2 середовище розробки Microsoft Visual Studio 2019.
4. Перелік питань, що потрібно опрацювати в роботі *вступ, аналіз проблемної області, обрання цільової моделі даних для міграції, аналіз існуючих методів і алгоритмів, постановка задачі, вибір стратегії міграції, математичне моделювання міграції БД, розробка алгоритму міграції, розробка програмного забезпечення, проектування реляційної схеми даних для експерименту, створення бази даних MS SQL, розробка запитів до бази даних, проведення дослідження, аналіз результатів дослідження, висновки.*

КАЛЕНДАРНИЙ ПЛАН

Номер	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналітичний огляд	24.01.2022 – 04.02.2022	виконано
2	Обрання цільової моделі даних для міграції, аналіз існуючих методів і алгоритмів	05.02.2022 – 15.02.2022	виконано
3	Постановка задачі	16.02.2022 – 25.02.2022	виконано
4	Математичне моделювання міграції БД, розробка алгоритму	26.02.2022 – 05.03.2022	виконано
5	Розробка програмного забезпечення	06.03.2022 – 15.03.2022	виконано
6	Проектування реляційної схеми даних, створення бази даних MS SQL	16.03.2022 – 25.03.2022	виконано
7	Розробка запитів до бази даних	26.03.2022 – 01.04.2022	виконано
8	Проведення експериментального дослідження	02.04.2022 – 12.04.2022	виконано
9	Аналіз результатів дослідження	13.04.2022 – 16.04.2022	виконано
10	Підготовка пояснювальної записки	17.04. 2022 – 30.04.2022	виконано
11	Підготовка презентації та доповіді	31.04.2022 – 05.05.2022	виконано
12	Нормоконтроль	11.05.2022	виконано
13	Рецензування	13.05.2022	виконано
14	Занесення диплома в електронний архів	13.05.2022	виконано
15	Попередній захист	14.05.2022	виконано

Дата видачі завдання 24 січня 2022р.

Студент _____
(підпис)

Керівник роботи _____ проф. Лєсна Н.С.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Кваліфікаційна робота магістра містить: 143 с., 37 рис., 22 табл., 26 джер.

БАЗА ДАНИХ, ГЕТЕРОГЕННА МІГРАЦІЯ, ГОМОГЕННА МІГРАЦІЯ, МОДЕЛЬ ДАНИХ, НЕРЕЛЯЦІЙНА МОДЕЛЬ, РЕЛЯЦІЙНА МОДЕЛЬ, ТЕОРІЯ МНОЖИН, MONGODB.

Об’єктом дослідження в роботі є міграція баз даних, моделі зберігання даних, процес перепроектування схеми даних з рамках міграції.

Предметом дослідження є методи підтримки гетерогенної модельно-неоднорідної міграції даних між реляційними та документо-орієнтовними моделями зберігання даних, технології розробки програмного забезпечення для автоматизації процесу міграції.

Мета роботи – дослідити методи підтримки міграції даних між реляційними і документними моделями даних, побудувати математичну модель та алгоритм для міграції даних, розробити програмну реалізацію алгоритму, порівняти ефективність розробленого алгоритму з уже існуючими.

Методи розробки базуються на платформі .NET Core 5 з мовою C#, СКБД MS SQL Server 2019, MongoDB 4.2, середовищі розробки Visual Studio 2019.

В роботі вирішуються наступні завдання: розгляд поняття та різновидів міграції даних, обґрунтування вибору документо-орієнтовної моделі даних в якості цільової, аналіз літературних джерел, що стосуються гетерогенної неоднорідно-модельної міграції даних, вибір підходу до формування алгоритму міграції даних, опис математичної моделі міграції даних за допомогою реляційної алгебри та теорії множин, представлення алгоритму міграції даних, тестування та дослідження ефективності розробленого алгоритму.

Отримано наступні результати: створено математичну модель методу гетерогенної модельно-неоднорідної міграції, яка передбачає перепроектування схеми на основі запитів, описано алгоритм застосування методу, порівняно ефективність цього методу з іншими методами, зроблено відповідні висновки.

DATABASE, DATA MODEL, HETEROGENEOUS MIGRATION, HOMOGENEOUS MIGRATION, MONGODB, NON-RELATIONAL MODEL, RELATIONAL MODEL, SET THEORY.

The object of research is the migration of databases, data storage models, the process of redesigning the data scheme within the migration.

The subject of the research is methods of support of heterogeneous model-inhomogeneous data migration between relational and document-oriented data storage models, technologies of software development for automation of migration process.

The aim of the work is to investigate the methods of supporting data migration between relational and documentary data models, to build a mathematical model and algorithm for data migration, to implement the algorithm programmatically, to compare the efficiency of the developed algorithm with existing ones.

Development methods are based on the .NET Core 5 platform with C # language, MS SQL Server 2019 DBMS, MongoDB 4.2, Visual Studio 2019 development environment.

The following tasks are solved in the work: consideration of the concept and types of data migration, substantiation of the choice of document-oriented data model as a target, analysis of literature sources related to heterogeneous inhomogeneous model data migration, choice of approach to data migration algorithm, description of mathematical data migration model using relational algebra and set theory, presentation of the data migration algorithm, testing and research of the efficiency of the developed algorithm.

The following results were obtained: a mathematical model of the method of heterogeneous model-inhomogeneous migration, which involves redesign of the scheme based on queries, described the algorithm of the method, compared the effectiveness of this method with other methods, made appropriate conclusions.

Я, Перетятко Марія Вікторівна, студентка гр. ІПЗм-20-2, здобувачка вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Дослідження методів підтримки міграції для різних моделей зберігання 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 ElAr KhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлена 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

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

БД – база даних;

СКБД – система керування базами даних;

CLI – command line interface;

GUI – graphical user interface;

JSON – javascript object notation;

MS SQL – Microsoft SQL;

NoSQL – not only SQL;

SQL – structured query language;

T-SQL – Transact-SQL.

ВСТУП

На сьогоднішній день у світі збільшується роль інформаційних технологій, з’являється все більше програмних застосунків, що охоплюють різноманітні сфери життя людей. Більшість програмних застосунків передбачають зберігання даних у тому чи іншому вигляді. Зі збільшенням ролі програмних систем розширюються масштаби їх використання, зростають об’єми необхідних для збереження даних та ускладнюється структура цих даних. Також розвиваються способи зберігання даних: створюються нові підходи до зберігання даних, нові типи баз даних, удосконалюються існуючі системи керування базами даних, з’являються гібридні бази даних, які уміщують в собі властивості та функції декількох інших баз даних тощо.

Традиційних баз даних часто недостатньо, щоб наздогнати зростаючі вимоги обробки великих даних. Звичайні реляційні бази даних мають обмеження моделі даних, архітектурні обмеження, недостатньо можливостей масштабованості та обмеження продуктивності. Рано чи пізно перед програмною системою може постати проблема неспроможності повноцінно функціонувати через обмеження використовуваної бази даних:

– при надмірному навантаженні на базу, яке досягає граничних значень при великій кількісті користувачів системи;
– при ускладненні бізнес-логіки і, як наслідок, складності використання моделі даних поточної бази даних для потреб цієї бізнес-логіки;
– при переході програмного застосунку до нового стеку технологій і технічній або логічній складності використання поточної бази даних з новим стеком технологій;
– при неможливості розвитку і конкурентоспроможності програмної системи на сучасному ринку за умови використання застарілої бази даних у цій системі (за законом Мура [1] приблизно кожні два роки відбувається значне зростання швидкості та можливостей технологій, це означає, що

задля підтримки конкурентоспроможності важливо весь час бути на крок попереду прогресу);

– при існуванні ризиків щодо повноцінної безпеки та цілісності даних на застарілих СКБД тощо.

Одним із способів вирішення вищезазначеної проблеми є перехід на іншу СКБД (більш нову, гнучку, з перевагами у можливостях, які є необхідними для певної програмної системи), при цьому існуючі дані мають бути перенесені на нову базу даних без втрат та пошкоджень і бути готовими до повноцінного функціонування в новій базі даних – цей процес називається міграцією.

Міграція бази даних є досить складним та тривалим процесом, оскільки вихідна база, та база, на яку проводиться міграція, можуть бути різнотипними, мати достеменно різні моделі зберігання даних, типи даних, способи роботи з даними, особливості функціонування (наприклад, міграції з реляційних баз даних до документо-орієнованих, подійних, графових тощо). Тому, щоб процес міграції пройшов вдало, необхідно мати чіткий план міграції, який включає в себе усі підготовчі дії, умови та заходи в рамках безпосередньої міграції та дії після завершення міграції даних (порядок переходу програмного застосунку до нової бази даних та ліквідування старої бази даних).

При виконанні кваліфікаційної роботи було проаналізовано предметну область міграції баз даних за допомогою сучасної наукової літератури, в тому числі і робіт фахівців кафедри ПІ (додаток А). На основі отриманих знань було сформовано постановку задачі, сплановано етап підготовки до експерименту та сам експеримент. Було створено презентацію, яка відображує основну інформацію щодо усіх етапів роботи (додаток В).

Метою роботи є побудова алгоритму для гетерогенної модельно-неоднорідної міграції даних, розробка програмного забезпечення для автоматизації міграції, порівняння ефективності розробленого алгоритму з уже існуючими алгоритмами.

Об’єктом дослідження в роботі є міграція баз даних, моделі зберігання даних, процес перепроектування схеми даних з рамках міграції.

Предметом дослідження є методи підтримки гетерогенної модельно-неоднорідної міграції даних між реляційними та документо-орієнтовними моделями зберігання даних, технології розробки програмного забезпечення для автоматизації процесу міграції.

Методи розробки а даній роботі базуються на платформі .NET Core 5 з мовою C#, вихідна база даних – СКБД MS SQL Server 2019, цільова база даних – MongoDB 4.2, середовищі розробки програмного коду Visual Studio 2019.

Наукова новизна кваліфікаційної роботи полягає у одержанні нового алгоритму міграції схеми даних з реляційної до документо-орієнтовної моделі на основі запитів до бази даних, а також у результатах тестування його ефективності у порівнянні з уже існуючими алгоритмами.

Одержаний алгоритм є готовим для використання при реальних міграціях баз даних задля перепроектування схеми даних з реляційної до документної моделі, використання є доцільним у випадку немалого об’єму вихідної бази даних.

Результати кваліфікаційної роботи були представлені на двох наукових конференціях, тези доповідей опубліковані у збірниках цих конференцій, також прийнята до друку одна стаття:

– XV Всеукраїнська науково-практична WEB конференція аспірантів, студентів та молодих вчених «Комп’ютерні інтелектуальні системи та мережі», тези представлені у додатку Е;

– Дванадцята міжнародна науково-технічна конференція «Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління», тези представлені у додатку Ж.

– рецензований науковий журнал відкритого доступу з переліку наукових фахових видань України (за спеціальністю 121 Інженерія програмного забезпечення) «Сучасний стан наукових досліджень та технологій в промисловості», матеріали статті представлені у додатку И.

1 АНАЛІЗ ПРОБЛЕМНОЇ ОБЛАСТІ

1.1 Аналітичний огляд

Міграція бази даних – це процес перенесення даних з однієї або кількох вихідних баз даних до однієї або кількох цільових баз даних визначеним методом [2]. Після завершення міграції повний, можливо реструктурований, набір вихідних даних міститься у цільових базах даних. Клієнти, які користувалися вихідними базами даних, переводяться на цільові бази даних, а вихідні бази даних не використовуються та можуть бути видалені у визначені терміни.

На рисунку 1 схематично зображено процес міграції бази даних (з трьох вихідних до двох цільових баз даних).

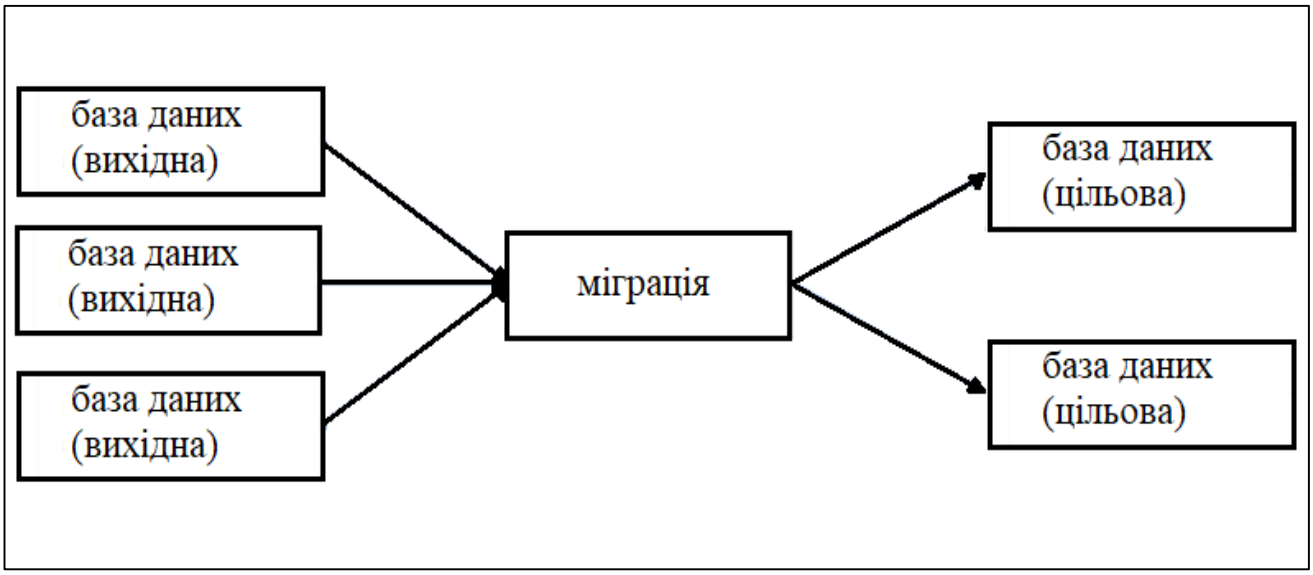

Рисунок 1 – Схема процесу міграції баз даних

У контексті технологій існує два види міграції баз даних:

– гомогенна;

– гетерогенна.

Гомогенна (однорідна) міграція – це міграція між базами даних, за якої вихідна і цільова бази належать до однієї і тієї ж технології баз даних [3],

наприклад, міграції з бази даних MySQL в базу даних MySQL, або з бази даних Oracle до бази даних Oracle. Однорідні міграції також включають міграції між системою баз даних, яка розміщена на власному сервері, наприклад, PostgreSQL, до її керованої версії, наприклад Cloud SQL (варіант PostgreSQL).

При гомогенній міграції схеми вихідної та цільової баз даних у більшості випадків ідентичні. Якщо схеми відрізняються, дані з вихідних баз даних повинні бути перетворені під час міграції.

Гетерогенна (неоднорідна) міграція баз даних – це міграція за якої вихідна і цільова бази належать до різних технологій баз даних [3], наприклад, міграція з бази даних MS SQL до MongoDB. Неоднорідна міграція бази даних може здійснюватися між однаковими моделями даних (наприклад, з реляційної на реляційну) або між різними моделями даних (наприклад, з реляційної на ключ-значення).

Міграція між різними технологіями баз даних не обов’язково передбачає різні моделі даних. Зокрема, Oracle, MySQL, PostgreSQL і Spanner підтримують реляційну модель даних. Однак багатомодельні бази даних, такі як Oracle, MySQL або PostgreSQL, підтримують декілька моделей даних. Наприклад, якщо багатомодельна база даних підтримує зберігання даних як документи JSON, то дані можуть бути перенесені в MongoDB без необхідності практичного перетворення, оскільки модель даних однакова у вихідній та цільовій базі даних.

Хоча відмінність між гомогенною та гетерогенною міграцією базується на технологіях баз даних, альтернативна категоризація базується на задіяних моделях баз даних.

Наприклад, міграція з бази даних Oracle до Spanner є модельно-однорідною міграцією, через те, що в обох базах використовується реляційна модель даних, тобто змінюється тільки технологія, яка використовується для бази даних. На рисунку 2 зображено схему гетерогенної модельно-однорідної міграції.

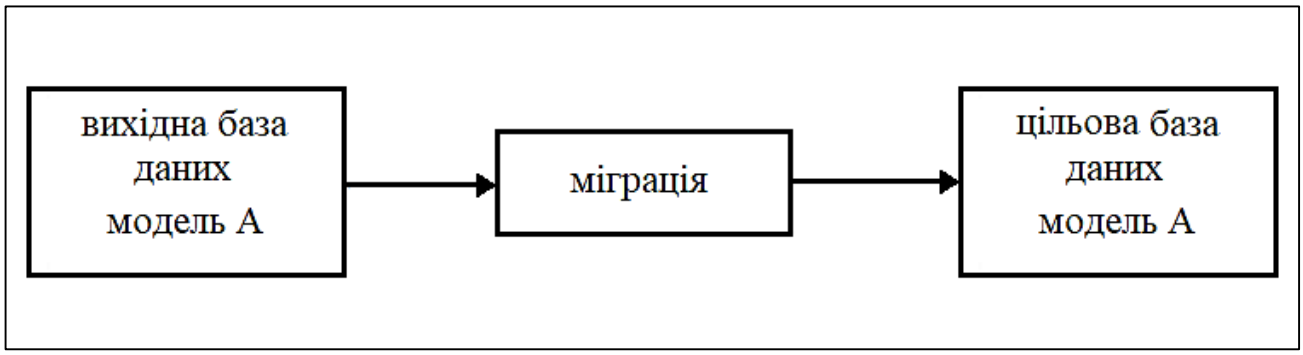

Рисунок 2 – Гетерогенна модельно-однорідна міграція баз даних

Міграція є модельно-неоднорідною при використанні у вихідних і цільових базах даних різних моделей зберігання даних, якщо, наприклад, дані, що зберігаються як об’єкти JSON в Oracle, перенесені до реляційної моделі в Spanner. На рисунку 3 зображено схему гетерогенної модельно-неоднорідної міграції.

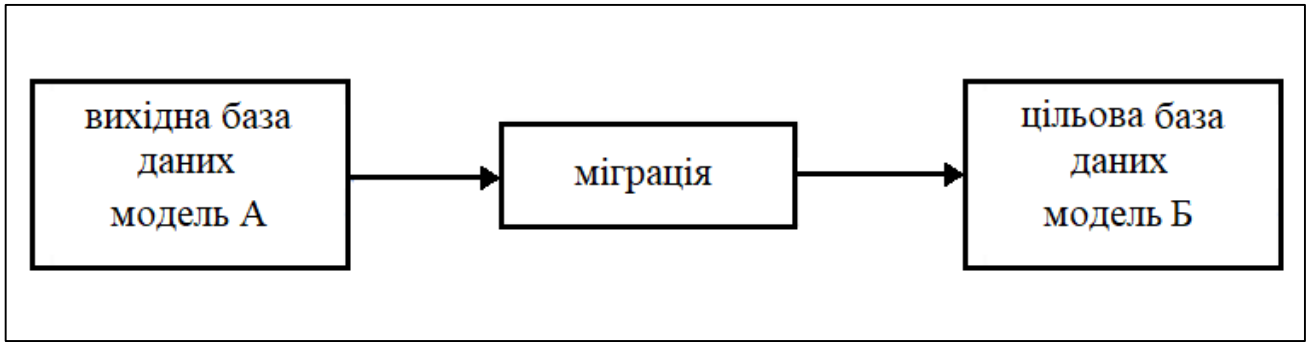

Рисунок 3 – Гетерогенна модельно-неоднорідна міграція баз даних

Розподіл по категоріях за моделлю даних більш точно у порівнянні з розподілом за системами баз даних відображає рівень складності переміщення даних.

Найбільш складним випадком міграції є той, за якого вихідна та цільова бази ґрунтуються на різних технологіях та в той же час мають різні моделі даних, саме такий випадок і буде у подальшому розглянуто і проаналізовано в рамках даної роботи.

По закінченню переміщення даних з вихідної бази у цільову треба перемикнути клієнтський доступ до цільової бази та утилізувати вихідну базу.

Процес клієнтського перемикання з вихідних до цільових баз даних складається з декількох частин [4]:

– для продовження міграції клієнти мають тимчасово закрити свої з’єднання з вихідними базами даних та під’єднатися до нових баз даних;
– закривши клієнтські з’єднання з вихідними базами даних відбувається процес перенесення даних, які залишилися у вихідних базах, до цільових баз. Цей процес дістав назву «draining» та виконується для гарантування того, що усі дані мігровано до нових баз даних;
– після перенесення даних слід перевірити функціонування цільових баз даних та клієнтські підключення.

Повна відсутність простою для клієнтів під час міграції даних неможлива, під час простоїв ймовірні випадки неможливості обробки запитів, і це ставить під загрозу роботу застосунку, тому однією з цілей під час міграції даних є мінімізація часу простою. Існують стратегії, за допомогою яких можливо зменшити час простоїв [5], а саме:

– запуск тестових клієнтів в режимі читання з цільовими базами даних на випередження, до початку міграції;
– аналіз та налаштування кількості переміщуваних даних при наближенні міграції, часткове переміщення даних визначеними порціями, загальний час міграції при цьому зросте, але тривалість простоїв зменшиться;
– підключення нових клієнтів до цільових баз даних при роботі старих клієнтів з вихідними базами даних, як наслідок, скорочення часу повного переходу до нових баз даних.

Існують декілька варіантів кардинальності міграцій баз даних:

– пряме відображення (1:1): дані однієї вихідної бази переміщують в одну цільову базу;
– консолідація (n:1): дані з декількох вихідних баз даних переміщуються до меншої кількості цільових баз даних, такий підхід може призвести до спрощення процедури управління базою даних;
– розподіл (1:n): дані з однієї вихідної бази переміщуються до визначеної (>1)

кількості цільових баз даних. Така кардинальність може бути використана, наприклад, при переміщенні вихідної централізованої бази даних з регіональними даними до декількох цільових регіональних баз даних;

– перерозподіл (n:m): дані з визначеної кількості вихідних баз даних переміщуються до визначеної кількості цільових баз даних. Така кардинальність є корисною у ситуації нерівномірної кількості даних у вихідних базах (і, відповідно, нерівномірного навантаження), внаслідок перерозподілу дані розподіляються між цільовими базами рівномірно.

Під час міграції даних можливо не тільки здійснити фактичне перенесення даних, але й перепроектувати базу даних [6], частіше архітектурні зміни впроваджують, якщо вихідні та цільові бази мають різні моделі, адже кожен тип моделі включає свої особливості та принципи, за яких модель буде працювати ефективніше, наприклад, при перенесенні даних з реляційної моделі на нереляційну можливо замість створення залежних таблиць перенести їх до вкладених таблиць (колекцій). Це зменшить кількість запитів до бази даних та скоротить час обробки пов’язаних сутностей.

1.2 Обрання цільової моделі даних для міграції

В контексті сучасних технологій реляційна модель зберігання даних має дві проблеми:

– робота з великими даними;

– робота з неструктурованими даними.

Вищезазначені проблеми знаходять своє рішення у базах даних з гнучкою структурою та більш ефективним способом роботи з великими обсягами даних – це NoSQL бази даних, що все частіше стають заміною реляційним базам даних і відіграють ключову роль для бізнес-застосунків в багатьох галузях. Такі бази даних оптимізовані для масштабування динамічних даних (наприклад тих, які

використовуються клієнтами в публікаціях у соціальних мережах).

Існує ряд проблем, пов'язаних із міграцією даних з реляційних до NoSQL баз даних. Для переміщення записів між моделями даних необхідно мати структуровану методологію для перетворення існуючих даних, а у випадку з міграцією реляційної моделі до нереляційної таких стандартів немає. Тому на теперішній час активно розроблюються різні стратегії та методи підтримки таких міграцій. Тому в межах даної роботи доцільно у якості цільової моделі даних, на прикладі якої буде здійснюватися розгляд подальшого матеріалу, обрати таку модель, що використовується в широко затребуваній на даний момент NoSQL базі даних.

Розглядаючи інформацію щодо сучасних найбільш популярних та широко використовуваних NoSQL баз даних [7] було встановлено, що під такі критерії підпадає документо-орієнтовна база даних MongoDB. У загальному рейтингу баз даних відповідно до ресурсу [8] MongoDB входить у топ-10, знаходячись на п'ятій позиції, та посідає перше місце в NoSQL рейтингу, оскільки усі попередні – реляційні бази даних (рисунок 4).

391 systems in ranking, April 2022

Rank Apr 2022	Rank Mar 2022	Rank Apr 2021	DBMS	Database Model	Score Apr 2022	Score Mar 2022	Score Apr 2021
1.	1.	1.	Oracle	Relational, Multi-model	1254.82	+3.50	-20.10
2.	2.	2.	MySQL	Relational, Multi-model	1204.16	+5.93	-16.53
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	938.46	+4.67	-69.51
4.	4.	4.	PostgreSQL	Relational, Multi-model	614.46	-2.47	+60.94
5.	5.	5.	MongoDB	Document, Multi-model	483.38	-2.28	+13.41
6.	6.	↑ 7.	Redis	Key-value, Multi-model	177.61	+0.85	+21.72
7.	↑ 8.	↑ 8.	Elasticsearch	Search engine, Multi-model	160.83	+0.89	+8.66
8.	↓ 7.	↓ 6.	IBM Db2	Relational, Multi-model	160.46	-1.69	+2.68
9.	9.	↑ 10.	Microsoft Access	Relational	142.78	+7.36	+26.06
10.	10.	↓ 9.	SQLite	Relational	132.80	+0.62	+7.74

Рисунок 4 – Рейтинг баз даних станом на квітень 2022 року [8]

Діаграма розподілу популярності баз даних за квітень 2022 року (рисунок 5) відповідно до ресурсу [8] показує, що документо-орієнтовні бази даних посідають

друге місце після реляційних, відсотковий показник реляційних становить 71,9 %, документних – 9,9 %.

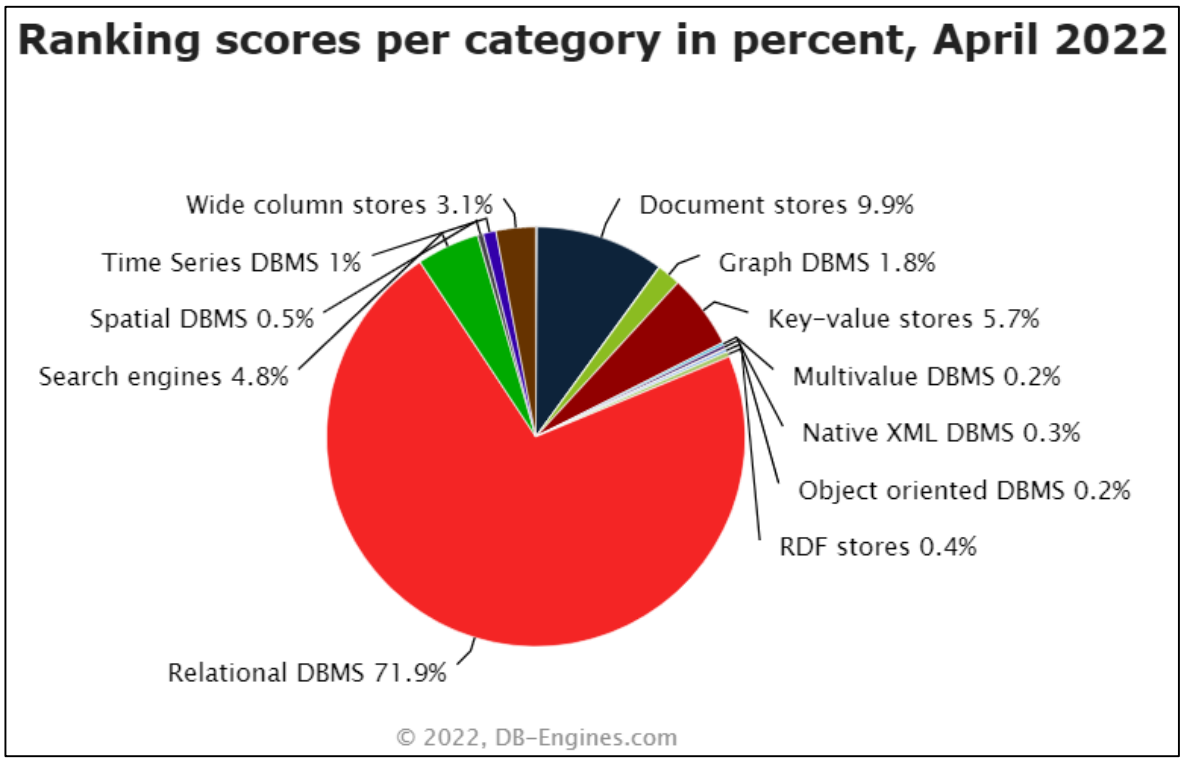

Рисунок 5 – Діаграма розподілу популярності баз даних за квітень 2022 року [8]

Також була проаналізована структура та можливості цієї бази даних: зокрема, існують ефективні технології шардінгу бази даних, формалізовані моделі керування великими даними [9; 10] тощо.

Велика кількість літератури присвячена порівнянню MongoDB з реляційними базами даних та детальна характеристика переваг першої [11]. На даний момент MongoDB застосовується в багатьох предметних областях (трьохмірної візуалізації, прогнозування використання теплової потужності будівель, контроль потоку ґрунтових вод і переносу забруднень, в системах моніторингу охорони здоров'я, IoT-застосунках тощо).

Отже, після вищенаведеного аналізу було прийняте рішення обрати документо-орієнтовну модель (на прикладі MongoDB) в якості NoSQL бази даних для використання у дослідженні.

1.3 Аналіз існуючих методів і алгоритмів

Питання міграції між базами даних, які представляються різними моделями є актуальним на сьогоднішній день, на даний момент воно вивчене не у достатній мірі для повної формалізації, тому дослідження, пов’язані з цим питанням, заслуговують на особливу увагу. Немає однозначної рекомендацій для виконання міграції тим чи іншим способом, і розробник сам має приймати рішення щодо методу міграції в залежності від характеристик даних та мети, для якої здійснюється міграція.

Розглянемо існуючі підходи і стратегії міграції з реляційної бази даних до MongoDB.

Один з найбільш простих способів переносу даних з реляційної моделі в MongoDB базується на тому, що більшість реляційних баз даних підтримують експорт таблиць в файли формату CSV. Навіть якщо така вбудована підтримка відсутня, можливо здійснити експорт за допомогою допоміжного програмного забезпечення і отримати файли в потрібному форматі. Після цього можна здійснити імпорт файлів в MongoDB за допомогою вбудованої команди [12]. Недоліком цього методу є те, що, по-перше, не враховуються існуючі зв’язки між таблицями, адже, по суті, отримуються лише списки за даними, а по-друге, кожна таблиця у реляційній базі відповідатиме колекції в MongoDB, не відбуватиметься логічної архітектурної перебудови таблиць до документо-орієнтовного стилю, це буде впливати на час виконання запитів, адже, як відомо, MongoDB гірше працює з запитами, які звертаються до багатьох колекцій документів. Тому такий метод доцільно використовувати лише для баз даних простої структури з невеликою кількістю таблиць і зв’язків.

Іншим підходом до міграції даних з реляційної до MongoDB є перенесення, яке складається з наступної послідовності дій [13 – 15]:

– вилучення даних з вихідної бази даних;

– робота з даними, приведення їх до потрібного вигляду (робота з типами

даних тощо);

– перенесення оброблених даних в цільову базу даних.

Недоліком цього підходу є те, що під час роботи з даними відбувається недостатнє приділення уваги схемам баз даних та зв’язкам між об’єктами, адже акцент ставиться саме на дані як такі, а не на їх структуризацію, тобто, як і в попередньому методі, під час використання такої стратегії не відбувається переструктуризація даних.

Ще одним відомим способом міграції даних з реляційної моделі до MongoDB є приведення даних на основі структури даних і запитів до даних [16]. Така міграція виконується в три етапи:

– опис структури реляційної бази даних, опис вимог до запитів до даних (у відповідності до бізнес-логіки);

– моделювання даних у запитно-орієнтовному контексті NoSql бази даних;

– моделювання схеми бази даних у запитно-орієнтовному контексті NoSql бази даних.

Недоліком цього способу, як і в попередніх випадках, є неврахування залежностей між об’єктами бази даних, оскільки нова структура бази даних враховує лише метадані про об’єкти і запити.

Наступним існуючим підходом до такого роду міграції є перенесення даних по правилах (шість правил), що описують три типи міграції: Column-Based, Document-Based та Graph-Based [17]. Правила описують кардинальність зв’язків між таблицями та спеціальні операції, які виконуються з однією з таблиць (операції агрегації тощо).

Недоліком цього підходу є те, що при міграції не враховується структура запитів до даних. Задля вирішення проблеми тривалості запиту, який звертається до декількох документів, цей підхід пропонує об’єднання усіх таблиць в одну NoSQL колекцію, але така дія неодмінно призведе до проблем з пам’яттю, так як об’єм такої колекції буде занадто великим. Застосування такого методу може бути виправданим лише у випадку, якщо база даних мала за обсягом.

Ще один із методів передбачає, що відносини в документо-орієнтовній

моделі можуть бути представлені у формі вбудованих документів і зв’язків між цими документами [18; 19]. Наприклад, у разі існування функціональної залежності між двома атрибутами, обидва атрибути буде трансформовано в один елемент даних в MongoDB. Той самий принцип застосовується і для часткової, і для транзитивної залежностей. Недоліком виступає те, що вбудовані документи можуть бути використані лише для обмеженого об’єму даних, а також немає чіткої ідентифікації форми цього вбудовування.

Існує ще один метод, він передбачає використання теорії нормалізації схеми баз даних та використання її у розробці схеми для MongoDB [20], але цей підхід не враховує зв’язків «багато-до-багатьох», первинні та зовнішні ключі.

Отже, після розгляду основних методів міграції даних між реляційною та json-подібною моделями даних, можна сказати, що розглянуті методи мають свої переваги та недоліки і повинні використовуватися в залежності від конкретної ситуації, можливо у комбінації один з одним.

2 ПОСТАНОВКА ЗАДАЧІ

Розглянувши предметну область і визначивши основні проблеми та перспективи міграції між реляційною та документо-орієнтовною моделями даних можна скласти перелік питань для розгляду та етапів їх виконання в рамках дослідження.

Перш за все, необхідно проаналізувати, оцінити плюси та мінуси існуючих методів підтримки міграцій між реляційними та документо-орієнтовними моделями даних, і на основі знань, отриманих під час розгляду та аналізу існуючих методів, розробити математичну модель та власне сам алгоритм для підтримки такого виду міграції.

Після цього необхідно розробити програмну систему, призначену для міграції баз даних за допомогою спроектованого на попередньому етапі алгоритму. При розробці програмної системи необхідно використовувати мову програмування C#, платформу .NET Core (тип застосунку Console Application).

Для тестування програмної системи необхідно обрати предметну область і для неї побудувати реляційну схему даних з різними типами зв’язків (з використанням СКБД MS SQL).

Далі необхідно проаналізувати особливості предметної галузі та визначити набір запитів різної складності, які відображують бізнес-цілі предметної області, тобто більш-менш наближено до реальності можуть бути необхідними/корисними в програмних системах, де така схема даних буде використовуватися.

Після підготовки схеми та запитів необхідно наповнити базу даних тестовою інформацією.

Маючи на вході тестову реляційну базу даних, розроблена програмна система має генерувати схему даних для документо-орієнтовної моделі даних MongoDB бази. Після закінчення генерації система повинна надавати можливість користувачеві переглядати згенеровану схему даних і приймати рішення щодо того, чи влаштовує його такий варіант схеми даних, користувач на цьому кроці

повинен бути забезпечений можливістю вносити корективи у згенеровану схему даних. Після підтвердження схеми даних застосунок має запустити процес міграції даних з MS SQL до MongoDB. По завершенню міграції необхідно впевнитися у тому, що усі дані перенеслися успішно, без втрат і пошкоджень, це можна перевірити шляхом співставлення кількості даних в таблицях та шляхом виконання аналогічних запитів на вихідній та цільовій базах даних, при співпадінні результатів можна стверджувати, що міграція пройшла успішно.

Для того, щоб порівняти швидкість міграції та швидкість виконання запитів на схемі даних, яка була згенерована спроектованим у рамках дослідження алгоритмом, необхідно обрати кілька існуючих методів міграції, завантажити в програмну систему схеми даних, які мають бути отримані з вихідної бази даних за цими методами, та запустити міграцію даних. Для порівняння швидкості міграції необхідно заміряти час міграції кожним з методів, для запитів – порівняти час виконання однакових запитів для кожної з цих баз даних.

Для того, щоб найбільш повно оцінити ефективність виконання запитів для різних методів міграції та зробити висновки щодо доцільності використання того чи іншого методу необхідно проводити тестування на різних об’ємах даних (різна ступінь заповненості бази даних), а також на різній кількості запитів. Тоді можливо буде порівняти використані методи та проаналізувати доцільність кожного з методів в неоднакових умовах.

Результати дослідження мають будуватися на залежностях часу виконання запита на різних об’ємах даних для усіх методів міграції схеми даних, взятих до розгляду в дослідженні.

Отже, в рамках дослідження потрібно вирішити наступні задачі:

– розглянути існуючі методи та механізми міграцій між різними моделями даних;
– проаналізувати існуючі методи та обрати методи для дослідження;
– розробити на основі отриманих знань власний метод підтримки міграції: описати математичну модель та спроектувати алгоритм;
– створити програмне забезпечення, необхідне для дослідження,

використовуючи мову програмування C#, платформу .NET Core (тип застосунку Console Application);

– провести підготовку до дослідження (створити схему бази даних, набір запитів до бази даних, заповнити базу даних тестовою інформацією);
– виконати міграції даних кожним із вибраних для дослідження методом та зафіксувати результати;
– виконати запити у цільовій базі даних з різними схемами, зафіксувати час виконання при різній кількості даних;
– проаналізувати отримані результати, зробити висновки/рекомендації щодо доцільності того чи іншого методу в певних умовах.

3 МОДЕЛЮВАННЯ МІГРАЦІЇ

3.1 Вибір методів та стратегій міграції

Під час вивчення та аналізу методів перетворення реляційної моделі даних до моделі даних MongoDB для подальшого розгляду і дослідження було визначено три основні загальні стратегії міграції:

– під час міграції для кожної таблиці реляційної бази даних створюється відповідна колекція в MongoDB [21];
– під час міграції усі таблиці реляційної бази даних об’єднуються в єдину колекцію даних MongoDB;
– під час міграції відбувається перепроектування схеми бази даних таким чином, щоб вона найбільш повно відповідала запитам до бази даних (як наслідок, спрощувала б виконання цих запитів).

Перші дві стратегії є зрозумілими та однозначними і не потребують використання допоміжних методів для реалізації.

Третя стратегія є більш складною і передбачає використання допоміжних методів, за допомогою яких будуть формуватися схеми колекцій в MongoDB з урахуванням запитів.

Одним з підходів при перетворенні моделей даних є використання реляційної алгебри і теорії множин [22]. При цьому здійснюється адаптування методів реляційної алгебри і теорії множин задля зручного використання у контексті моделей і схем баз даних. За допомогою такого підходу можливо, користуючись формальною мовою теорії множин, розробити конкретні необхідні кроки перетворення моделей та скласти загальний алгоритм підтримки гетерогенної модельно-неоднорідної міграції для обраних моделей даних.

Використання реляційної алгебри і теорії множин є доцільним для реалізації стратегії перепроектування схеми бази даних у відповідність до запитів, адже:

– ієрархію будови моделей та запити досить зручно представляти множинами (простими або складеними);

– множини зручно аналізувати;

– є загальновідомі правила та операції для роботи з множинами [23], отже досить зручно описувати і виконувати дії з множинами та представляти результати цих дій.

Тому саме такий підхід буде використано в якості інструменту для подальшого математичного моделювання методу підтримки міграції.

Програмне забезпечення в рамках даної роботи буде розроблено за допомогою кросплатформеного фреймворку ASP.NET Core (у порівнянні з ASP.NET цей фреймворк має більшу потужність), відповідно, на мові програмування C#, тип для проекту – консольний застосунок. Розробка буде здійснюватися у середовищі програмування Visual Studio 2019, оскільки воно надає широкі можливості для розробки.

3.2 Математичне моделювання міграції БД

Представимо вхідну реляційну схему бази даних за допомогою теорії множин [24 – 26]. Нехай T – це множина усіх таблиць в реляційній схемі, множина T складається з елементів T_i, де r – це номер таблиці в схемі, представлення у вигляді формули виглядає наступним чином (формула 1):

$$T = \{T_i, i = 1, \dots n\}, \tag{1}$$

де T_i – i-та таблиця у множині таблиць T;

i – номер таблиці у множині таблиць;

n – загальна кількість таблиць у схемі.

Представлення множини таблиць реляційної бази даних наведено на рисунку 6.

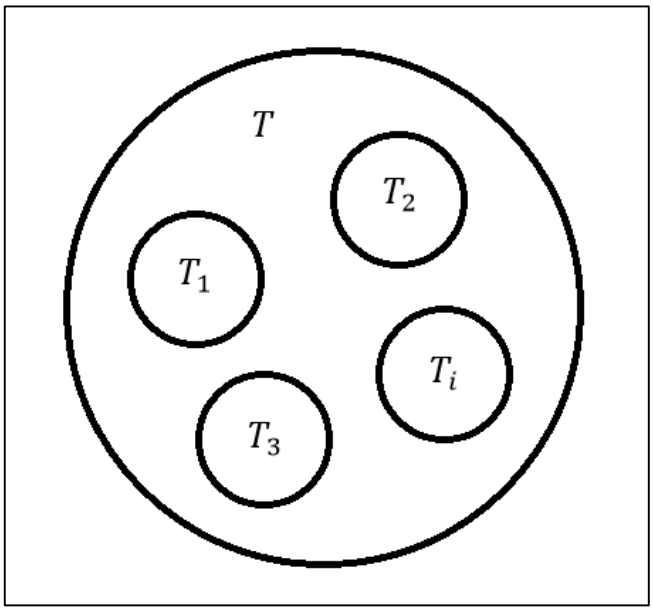

Рисунок 6 – Представлення схеми реляційної бази даних у вигляді множин

Кожна таблиця складається з полів, тобто таблиця T_i являється множиною полів F_{ij} (формула 2):

$$T_i = \{F_{ij}, i = 1, \dots n;\ j = 1, \dots i_k\}, \tag{2}$$

де F_{ij} – j-те поле у i-тій таблиці;

i – номер таблиці у множині таблиць;

n – загальна кількість таблиць у схемі;

j – номер поля у i-тій таблиці;

i_k – загальна кількість полів у i-тій таблиці.

Представлення схеми реляційної бази даних у вигляді множин наведено на рисунку 7.

Нехай Q – це множина усіх запитів до бази даних, які передбачені бізнес-логікою застосунку (формула 3):

$$Q = \{Q_l, l = 1, \dots m\}, \tag{3}$$

де Q_l – l-тий запит у множині запитів Q;

l – номер запиту у множині запитів;

m – загальна кількість запитів у множині запитів.

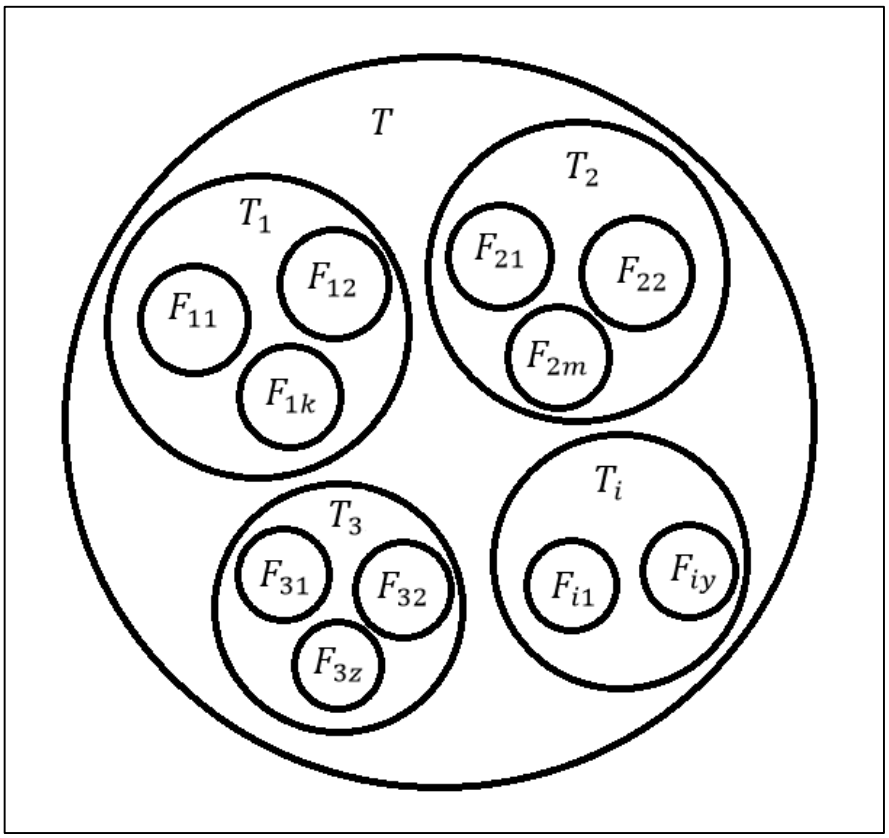

Рисунок 7 – Представлення схеми реляційної бази даних у вигляді множин

В свою чергу кожен із запитів звертається до деякої підмножини усіх полів бази даних, тобто кожний запит можна представити як наступну множину полів (формула 4):

$$Q_l = \{F_{ij}, i \le n;\ j \le i_k; l = 1, \dots x\}, \quad (4)$$

де F_{ij} – j-те поле у i-тій таблиці;

i – номер таблиці у множині таблиць;

n – загальна кількість таблиць у схемі;

j – номер поля у i-тій таблиці;

i_k – загальна кількість полів у і-тій таблиці;

х – загальна кількість запитів.

Представлення запитів до бази даних у вигляді множин наведено на рисунку 8.

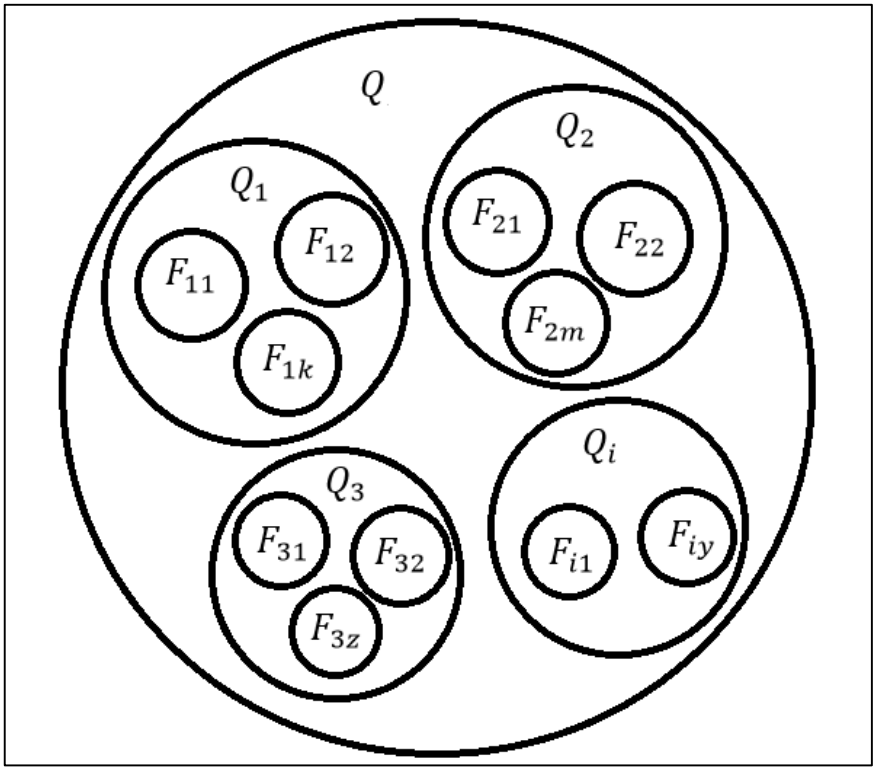

Рисунок 8 – Представлення множини запитів до бази даних

Для кожного поля необхідно знайти множину запитів, у яких це поле використовується (формула 5):

$$F_{ij}^{q} = \{Q_l, l = 1, \dots z; z \leq x | F_{ij} \epsilon Q_l\}, \quad (5)$$

де F_{ij}^{q} – множина запитів, у яких використовується поле F_{ij};

Q_l – l-тий запит, в якому використане поле F_{ij};

z – кількість запитів, в яких використане поле F_{ij};

x – загальна кількість запитів.

На початку перетворення моделей необхідно вилучити ті поля, які не використовуються в жодному запиті, і помістити такі поля в окрему колекцію. Такі поля відповідають наступній умові (формула 6):

$$\left|F_{ij}^{q}\right| = 0; \tag{6}$$

Усі такі поля можуть бути включені до одної (або декількох) колекції C_1 ($C_1 \dots C_k$), у множинному представленні ця колекція виглядає наступним чином (формула 7):

$$C_1 = \{F_{ij}, i \le n;\ j \le i_k \left|\left|F_{ij}^{q}\right| = 0\right\}, \tag{7}$$

де F_{ij} – j-те поле у i-тій таблиці;

i – номер таблиці у множині таблиць;

n – загальна кількість таблиць у схемі;

j – номер поля у i-тій таблиці;

i_k – загальна кількість полів у i-тій таблиці.

На цьому етапі декілька колекцій створюється в тому випадку, якщо ті поля, які потрапляють у вищезазначену категорію ($\left|F_{ij}^{q}\right| = 0$) ніяк не пов’язані між собою та з точки зору логічності представлення схеми даних не можуть бути об’єднані в одну колекцію.

Далі необхідно вибрати ті поля, які беруть участь тільки в одному запиті (формула 8):

$$\left|F_{ij}^{q}\right| = 1; \tag{8}$$

Поля, які задовольняють вищенаведеній умові входять до нової колекції C_z (формула 9):

$$C_z = \left\{F_{ij}, i \le n;\ j \le i_k \,\middle|\, F_{ij} \epsilon Q_l \ \&\ \left|F_{ij}^{q}\right| = 1\right\}, z > 1, \tag{9}$$

де F_{ij} – j-те поле у i-тій таблиці

n – загальна кількість таблиць у схемі;

i_k – загальна кількість полів у i-тій таблиці;

Q_l – l-тий запит;

F_{ij}^q – множина запитів, у яких використовується поле F_{ij};

z – номер колекції.

Так само, як і для попередньої множини, може створюватися не одна, а декілька колекцій, якщо ті поля не пов'язані мж собою та з точки зору логіки не можуть бути об'єднані в одну колекцію.

На цьому етапі прибираємо з розгляду ті поля, які вже використано в попередніх колекціях.

Складаємо множину запитів H, з якою будемо працювати далі, шляхом вилучення із загальної множини усіх запитів ті запити, усі поля яких вже належать до попередньо знайдених колекцій.

Складаємо парні перетини множин $Q_l \epsilon H$, ці перетини мають наступний вигляд (формула 10):

$$Q'_k = Q_i \cap Q_j; j \neq i; i, j = 1, \dots |H|, \tag{10}$$

де Q'_k – k-тий попарний перетин;

H – множина запитів, що розглядаються;

Q_i – i-та множина запитів;

Q_j – j-та множина запитів.

Отримані таким чином непусті множини утворюють нову множину M. За умови, якщо $H - M \neq \emptyset$, то нова колекція складатиметься з тих полів, що входять до цієї різниці (формула 11):

$$C_g = \left\{F_{ij}, i \leq n;\ j \leq i_k \,\middle|\, F_{ij}\epsilon(H - M)\right\}, \tag{11}$$

де C_g – нова колекція;

H – множина запитів, що розглядаються;

M – множина непорожніх перетинів;

F_{ij}– поле з різниці $H - M$.

З множини взятих до розгляду полів видаляються ті, що увійшли до вищезнайденої множини. Після отримання колекції перевизначемо множину H задля продовження алгоритму (формула 12):

$$H = M, \tag{12}$$

де H – множина запитів, які розглядаються;

M – множина непорожніх перетинів.

Тепер знаходимо перетин полів запитів, але вже по 3 елементи (формула 13):

$$Q_k'' = Q_i \cap Q_j \cap Q_m; j \neq i; i, j, m = 1, \dots |H|, \tag{13}$$

де Q_k'' – k-тий перетин по три елементи;

H – множина запитів, що розглядаються;

Q_i – i-та множина запитів;

Q_j – j-та множина запитів;

Q_m – m-та множина запитів.

Аналогічно, як і для двох елементів, отримані непусті множини формують нову множину M і якщо $H - M \neq \emptyset$, то утворюється нова колекція (формула 11). Таким чином алгоритм повторюється до тих пір, поки на певному кроці не залишиться лише один перетин, тобто множина H не буде складатися з одного елементу, при досягненні цієї умови необхідно сформувати нову множину, що складатиметься з полів, які увійшли до останнього перетину та не увійшли до всіх попередніх колекцій.

Таким чином у результаті роботи методу буде сформовано сукупність колекцій документів зі своїми множинами полів.

У подальших дослідженнях планується на реальних прикладах баз даних

порівняти такий алгоритм для міграції між моделями даних з іншими алгоритмами та дослідити його ефективність.

3.3 Розробка алгоритму

На рисунку 9 представлено розроблений алгоритм за допомогою діаграми активностей.

Підготовчим кроком для алгоритму міграції бази даних за реляційної моделі до документо-орієнтовної моделі на основі запитів є визначення набору запитів до бази даних. Пропонується використати один з варіантів визначення запитів для бази даних:

– для комерційних проектів у більшості випадків окрім самої бази схеми відомі також запити до цієї бази, якими оперує застосунок, в цьому випадку для алгоритму беруться відомі запити;

– за умови, якщо набір запитів невідомий, доцільним є більш глибокий аналіз предметної області та схеми бази даних і самостійне складання тестових запитів, які з найбільшою вірогідністю покриватимуть бізнес-логіку предметної області, при цьому відповідатимуть призначенню застосунку, де ця база даних використовується або може використовуватися;

– якщо попередні два варіанти не можливо використати в певній ситуації, то в якості запитів пропонується розглядати зв’язки між таблицями.

Після визначення набору запитів необхідно, використовуючи теорію множин, попрацювати з представленням таблиць та запитів для подальшої роботи алгоритму, а саме:

– представити кожну таблицю як множину полів;

– представити кожен запит, як множину полів, які використовуються в ньому (у будь-якій частині: вибірка, групування тощо);

– представити кожне поле як множину запитів, в яких воно бере участь.

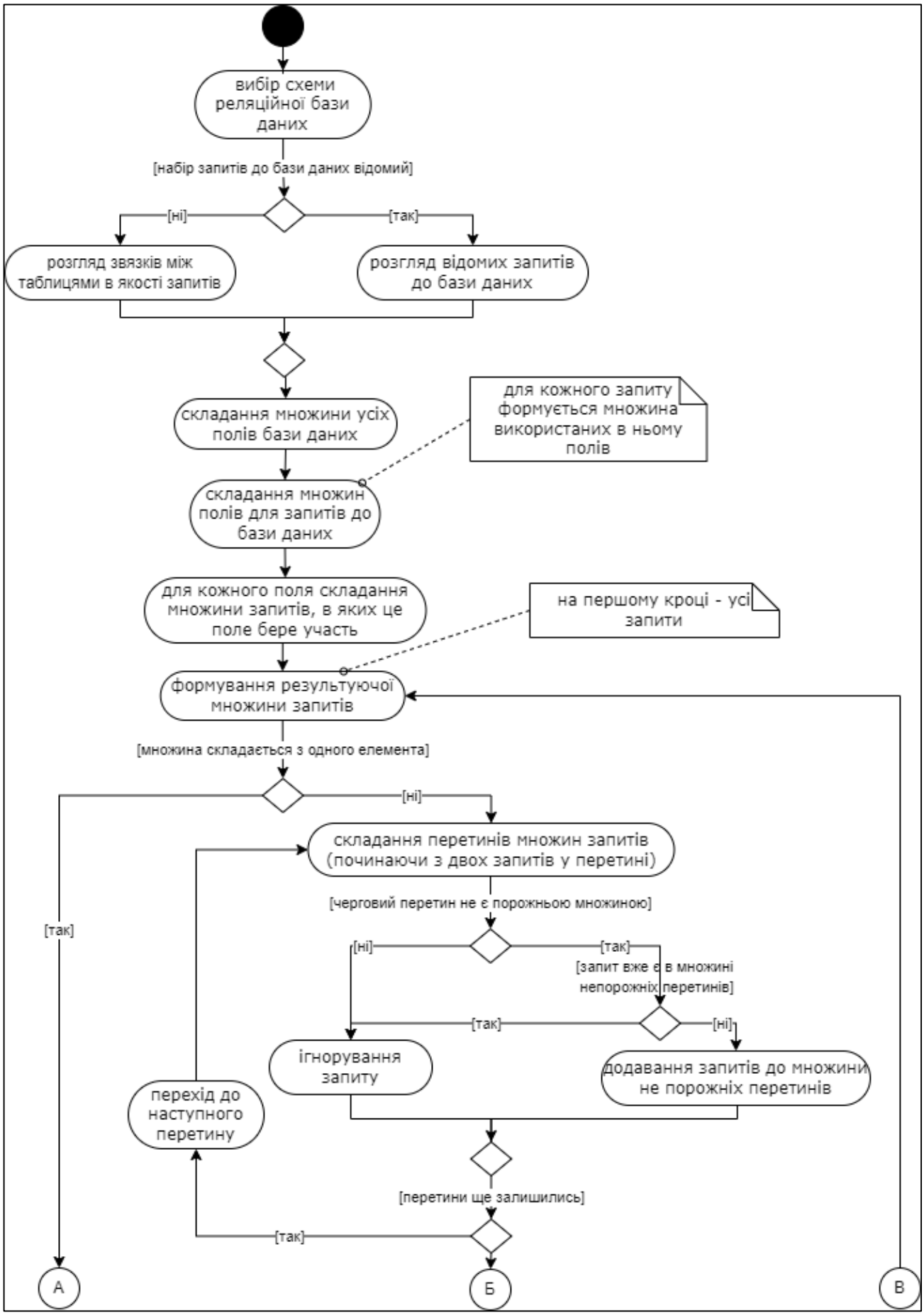

Рисунок 9 – UML-діаграма активності

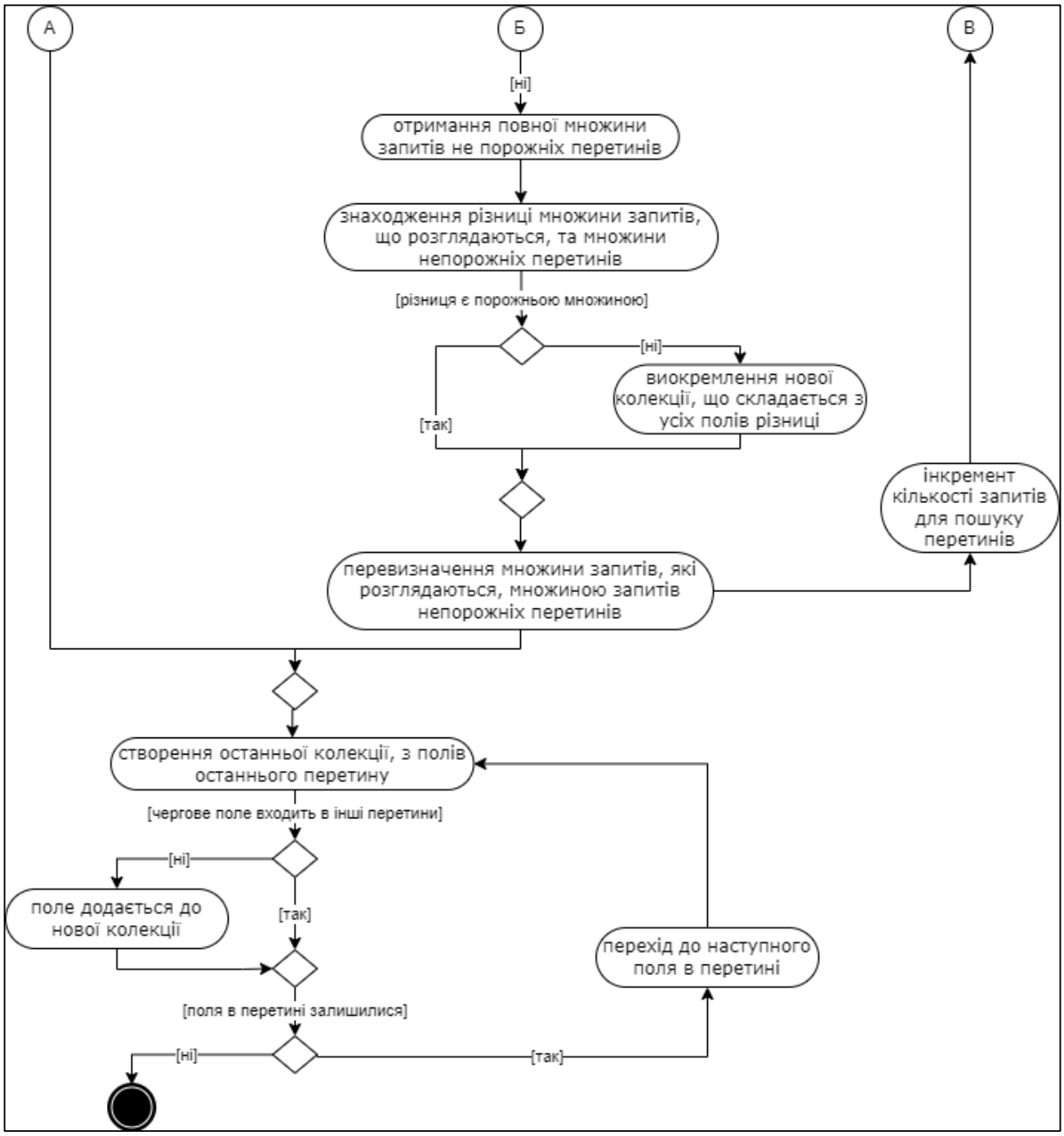

Рисунок 9, аркуш 2

Якщо на цьому етапі виявляється присутність таких полів, які не беруть участь у жодному із запитів, необхідно об’єднати їх в одну колекцію (або декілька колекцій, якщо бізнес-логіка не дозволяє об’єднати ці поля в одну колекцію) з урахуванням зв’язків між початковими таблицями, можливо з різними ступенями вкладеності.

Наступним етапом алгоритму є складання перетинів множин полів в запитах,

починаючи з попарних перетинів. Для кожного запиту, тобто множини його полів, має бути знайдений попарний перетин з усіма іншими множинами полів запитів. Якщо черговий перетин не є порожньою множиною, то необхідно додати запити з перетину до множини непорожніх перетинів. Таким чином здійснюється прохід по всіх попарних перетинах і запити з чергового перетину або додаються, або не додаються до множини непорожніх перетинів (якщо певний запит вже міститься в результуючій множині непорожніх перетинів, до вдруге його додавати не потрібно).

По закінченню формування результуючої множини непорожніх перетинів необхідно знайти різницю множини усіх запитів, які розглядалися на цьому етапі (при попарних перетинах – уся множина запитів) та множини запитів, які увійшли в множину непорожніх перетинів. Якщо знайдена різниця не є порожньою множиною, то отримана різниця складає нову колекцію (при цьому необхідно додатково перевірити, що кожне з полів входить до цієї різниці один раз, якщо входжень більше одного, то видалити повторювальні поля). Також, як і при формуванні першої колекції, необхідно звернути увагу на зв’язки між початковими таблицями, та при необхідності сформувати різні ступені вкладеності новоствореної колекції.

Для наступного кроку необхідно перевизначити множину запитів, що розглядаються, множиною непорожніх перетинів запитів, отриманих на попередньому кроці.

Тепер з множини запитів, які розглядаються на поточному кроці, необхідно скласти перетини полів запитів, але вже інкрементуючи кількість запитів для перетинів – тобто, на даному кроці, по три запити в кожному перетині. Аналогічно до дій з попарними перетинами, необхідно визначити множину запитів з непорожніх перетинів (без повторень елементів у результуючій множині). Отримавши результуючу множину непорожніх перетинів, необхідно знайти різницю множин запитів, що розглядаються на даному кроці, і поточної множини непорожніх перетинів. Якщо знайдена різниця не є порожньою множиною, то отримана різниця складає нову колекцію (з урахуванням зв’язків між початковими

таблицями, бізнес-логікою, оформивши необхідні рівні вкладеності у колекції тощо).

Аналогічні дії по формуванню нових колекцій, з інкрементом кількості запитів для перетину на кожному кроці, повторюються до тих пір, поки в множині запитів, які розглядаються на певному кроці, залишиться один елемент – тобто один запит, і так як з цим запитом вже не є можливим побудова перетину, відбувається перехід до наступного етапу алгоритму.

На цьому етапі перевіряється кожне поле з множини полів цього запита та за умови, якщо це поле ще не належить жодній зі створених протягом попередньої роботи алгоритму колекцій, воно додається в останню колекцію.

Таким чином, по завершенню роботи алгоритму усі поля є розподіленими між колекціями.

4 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Кодування програмного забезпечення

В рамках дослідження було розроблено програмне забезпечення, яке виконує дві задачі для підтримки міграції з реляційної моделі даних до документо-орієнтовної моделі даних:

– виконує міграцію схеми даних, що включає в себе перепроектування схеми за алгоритмом підтримки міграції моделей на основі запитів до даних, спроектованим в даному дослідженні;

– виконує міграцію даних, тобто здійснює перенесення інформації з однієї бази даних в іншу.

Ці дві частини системи можуть працювати окремо одна від одної, тобто є незалежними, наприклад, може виконуватися лише одна з них: у випадку, якщо вже є готова схема даних для документо-орієнтовної моделі, необхідно запустити лише другу функціональну частину програмного забезпечення – фактичне перенесення даних, або навпаки, якщо перенесення даних в певний момент часу не потрібно, але є необхідність отримати схему даних, то необхідно запустити лише першу функціональну частину програмного забезпечення.

Програмна система розроблена за допомогою наступних технологій: платформи .NET Core, мови програмування C#, використано тип застосунку Console Application.

З метою представлення об’єктів для підтримки міграції була розроблена наступна модель класів:

– клас «Scheme» для представлення схеми даних, що містить назву схеми та множину таблиць цієї схеми:

```
public class Scheme
{
    public String Name { get; set; }
    public List<Table> TablesSet { get; set; }
}
```

– клас «Table» для представлення таблиці з бази даних, що містить назву таблиці та множину полів таблиці:

```
public class Table
{
    public String Name { get; set; }
    public List<Field> FieldsSet { get; set; }
}
```

– клас «Field» для представлення поля, містить назву поля, відомості щодо того, чи є поле первинним або зовнішнім ключем, значення цього поля та тип даних:

```
public class Field
{
    public String Name { get; set; }
    public Boolean IsPrimary { get; set; }
    public String Value { get; set; }
    public String Type { get; set; }
    public Table Table { get; set; }
}
```

– клас «Query» для представлення запиту, містить назву запиту та набір полів, які в цьому запиті використовуються:

```
public class Query
{
    public String Name { get; set; }
    public List<Field> FieldsSet { get; set; }
}
```

Генерація схеми даних для документо-орієнтовної моделі починається так: на вхід програми подаються скрипти створення таблиць (CREATE TABLE), та список запитів до бази даних, програма виконує парсинг схеми даних відповідно до цих скриптів та створює необхідні вищезазначені об’єкти.

Далі запускається власне алгоритм перепроектування схеми, який було спроектовано у даному дослідженні. Наведемо основні елементи цього алгоритму

в програмній інтерпретації.

Програмний код, який перевіряє, чи використовуються поля певної таблиці в запитах, виглядає наступним чином (на вхід методу подаються таблиця та список запитів, які розглядаються на поточному кроці, метод повертає список полів, які не використовуються в жодному запиті):

```
public static List<Field> FindUnusedFields(Table table, List<Query>
queriesSet)
{
    List<Field> unusedFields = new List<Field>();

    foreach (Field tableField in table.FieldsSet)
    {
        Boolean isFieldUsed = false;
        foreach (Query query in queriesSet)
        {
            if (query.FieldsSet.Contains(tableField))
            {
                isFieldUsed = true;
                break;
            }
        }
        if (!isFieldUsed)
        {
            unusedFields.Add(tableField);
        }
    }

    if (unusedFields.Count > 0 && unusedFields.Where(x =>
x.IsPrimary).Count() == 0)
    {
        unusedFields.AddRange(table.FieldsSet.Where(x => x.IsPrimary));
    }

    if (unusedFields.Count > 0 && unusedFields.All(x => x.IsPrimary))
    {
        unusedFields.Clear();
    }

    return unusedFields;
}
```

Після цього з тих полів, які не беруть участь в жодному запиті, формується окрема колекція.

Програмний код, який визначає поля, які використовуються лише в одному запиті, виглядає наступним чином (на вхід методу подаються таблиця та список

запитів, які розглядаються на поточному кроці, метод повертає список полів, які використовуються лише в одному запиті):

```
public static List<Field> FindOnceUsedFields(Table table, List<Query>
queriesSet)
{
    List<Field> onceUsedFields = new List<Field>();
    foreach (Field tableField in table.FieldsSet)
    {
        Int32 fieldUsedCount = 0;
        foreach (Query query in queriesSet)
        {
            if (query.FieldsSet.Contains(tableField))
            {
                ++fieldUsedCount;
            }
        }

        if (fieldUsedCount == 1)
        {
            onceUsedFields.Add(tableField);
        }
    }

    if (onceUsedFields.Count > 0 && onceUsedFields.Where(x =>
x.IsPrimary).Count() == 0)
    {
        onceUsedFields.AddRange(table.FieldsSet.Where(x => x.IsPrimary));
    }

    return onceUsedFields;
}
```

Після цього з тих полів, які беруть участь лише в одному запиті, формується окрема колекція.

Однією з основних логічних частин алгоритму є пошук перетинів множин, код для пошуку перетинів наступний (на вхід подається список запитів, які розглядаються, та кількість запитів для перетинів, в якості результату метод повертає список тих запитів, які в перетині з іншими утворюють не порожню множину, результат додатково фільтрується задля відсутності дублюючих елементів):

```
public static List<Query> FindQueriesWithIntersections(IEnumerable<Query>
queriers, Int32 numberOfQueries)
```

```
{
    IEnumerable<IEnumerable<Query>> permutations =
GetPermutations(queriers, numberOfQueries);
    List<Query> intersectedQueries = new List<Query>();

    foreach (var permutation in permutations)
    {
        List<Field> commonList = new
List<Field>(permutation.ToList()[0].FieldsSet);
        foreach (Query query in permutation)
        {
            commonList = commonList.Intersect(query.FieldsSet).ToList();
        }

        if (commonList.Count > 0)
        {
            intersectedQueries.AddRange(permutation);
        }
    }

    return intersectedQueries.Distinct().ToList();
}
```

Для попереднього фрагменту коду було написано допоміжний метод для пошуку комбінацій без повторень, бо підмножини запитів по визначеній в параметрах методу кількості елементів є комбінаціями без повторень з елементів множини запитів, які розглядаються на даному кроці. Програмний код для цього методу виглядає наступним чином (на вхід методу подається колекція даних та число елементів для комбінації, метод повертає двомірний список комбінацій елементів):

```
public IEnumerable<IEnumerable<T>> GetPermutations<T>(IEnumerable<T> items,
Int32 count)
{
    Int32 i = 0;

    foreach (T item in items)
    {
        if (count == 1)
        {
            yield return new T[] { item };
        }
        else
        {
            foreach (IEnumerable<T> result in GetPermutations(items.Skip(i
+ 1), count - 1))
            {
                yield return new T[] { item }.Concat(result);
            }
```

```
        }

        ++i;
    }
}
```

Програмний код, який відповідає за процес формування колекцій даних відповідно до знайдених множин непорожніх та порожніх перетинів представлено в додатку Г (на вхід подаються список запитів та список таблиць).

Після завершення алгоритму програмна система видає в результаті нову схему даних, зберігаючи її в текстовий файл, код для збереження результату наступний (на вхід подається згенерована схема даних та назва файлу для збереження результату):

```
public void SaveResultScheme(String fileName, Scheme scheme)
{
    if (!File.Exists(fileName))
    {
        using (StreamWriter writer = File.CreateText(fileName))
        {
            foreach (Table table in scheme.TablesSet)
            {
                writer.WriteLine(table.Name);
                writer.WriteLine("{");
                foreach (Field field in table.FieldsSet)
                {
                    writer.WriteLine($"Name:{field.Name};
Type:{field.Type};  IsPrimary:{field.IsPrimary}");
                }
                writer.WriteLine("}");
            }
        }
    }
}
```

Друга частина функціоналу програмної системи виконує функцію безпосереднього перенесення даних з MS SQL до MongoDB, тобто з вихідної бази у цільову.

Програма на вхід приймає схему даних для документо-орієнтовної моделі (з текстового файлу), парсить текстову інформацію в об’єкти. Після цього відповідно до полів кожної колекції у новоствореній документо-орієнтовній моделі дістаються дані з вихідної бази даних та додаються як документи до колекцій (зберігаючись

при цьому у цільовій базі даних).

Основний програмний метод для міграції даних виглядає наступним чином (на вхід методу подаються список таблиць, що відповідають документо-орієнтовній моделі, заповнених даними з реляційних таблиць, назва бази даних для MongoDB, назва колекції для MongoDB):

```
async Task MigrateData(List<Table> tables, String dbName, String
collectionName)
{
    MongoClient client = new MongoClient();
    MongoServer server = client.GetServer();
    MongoDatabase database = server.GetDatabase(dbName);
    MongoCollection collection =
database.GetCollection<BsonDocument>(collectionName);

    List<BsonDocument> documents = new List<BsonDocument>();

    foreach (Table table in tables)
    {
        BsonDocument document = new BsonDocument();

        foreach (Field field in table.FieldsSet)
        {
            Object value;

            switch (field.Type)
            {
                case ("Int"):
                    value = Int64.Parse(field.Value);
                    break;
                case ("Double"):
                    value = Double.Parse(field.Value);
                    break;
                case ("Datetime"):
                    value = Double.Parse(field.Value);
                    break;
                default:
                    value = field.Value;
                    break;
            }
            document.Add(field.Name, value.ToBson());
        }

        documents.Add(document);
    }
}
```

Після виконання вищенаведеного методу для усіх даних кожної з таблиць створено базу даних MongoDB з усіма необхідними колекціями відповідно до

цільової схеми даних, новостворені колекції заповнені даними, перенесеними з MS SQL.

4.2 Тестування програмного забезпечення

Після завершення етапу кодування було проведено перевірку працездатності розробленого програмного забезпечення. Спочатку було проведено статичне тестування вихідної програми, тобто вихідний код програми було проаналізовано на відповідність загальноприйнятим нормам і принципам програмування на мові C#, виправлено неточності.

З метою того, щоб впевнитися у працездатності програмного забезпечення, для методів, що вміщують у собі основні логічні частини алгоритму, було написано unit-тести, розроблені за допомогою засобів бібліотеки для модульного тестування XUnit.

На рисунку 10 зображено результати проходження unit-тестів.

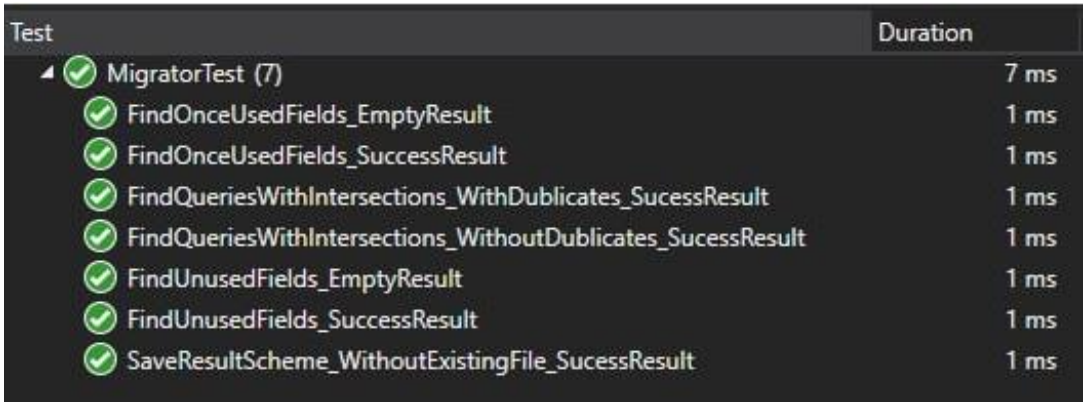

Рисунок 10 – Результати unit-тестування програмного забезпечення

Усі розроблені тести пройшли успішно, тобто на цьому етапі можна стверджувати, що програмне забезпечення працює коректно та готове для використання на реальних даних.

4.3 Опис інтерфейсу програмного забезпечення

На даний момент програмне забезпечення реалізоване у вигляді консольного додатку. Консольне вікно зображує вхідні дані (кількість реляційних таблиць та запитів), виводить усі згенеровані колекції з полями (рисунок 11).

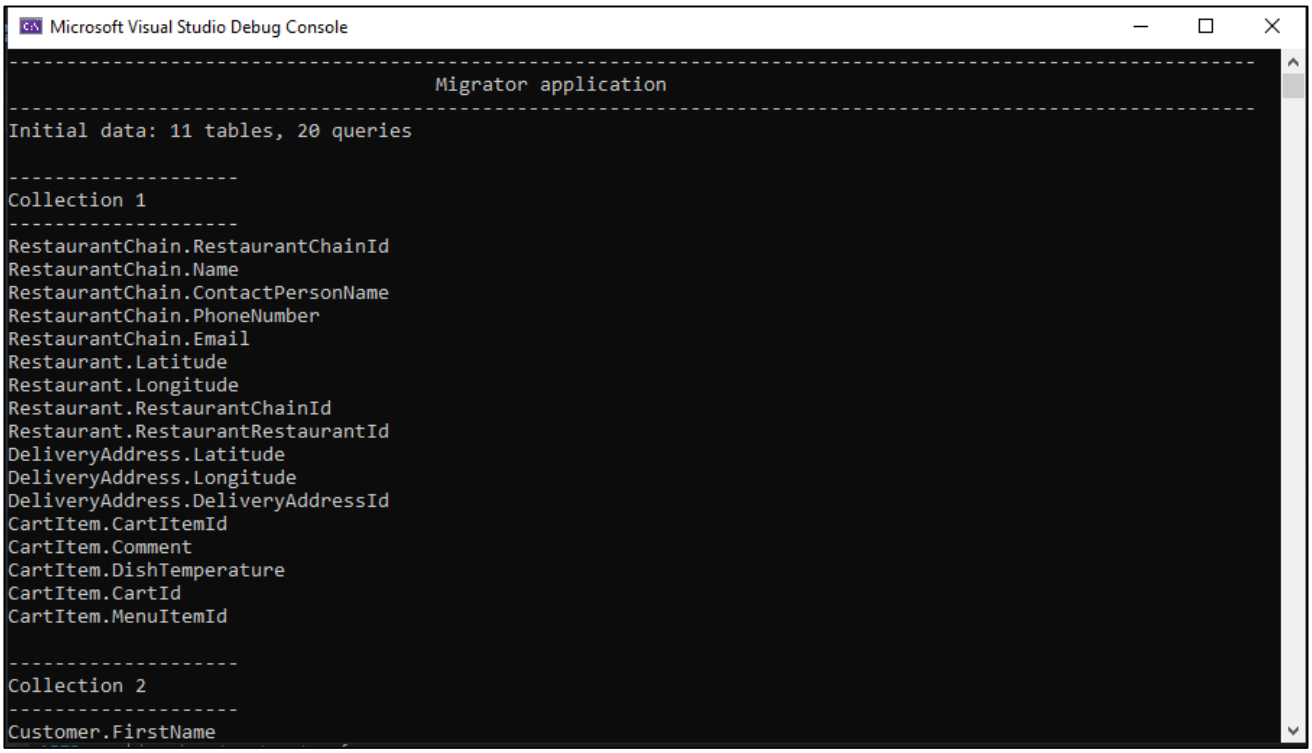

Рисунок 11 – Консольний інтерфейс застосунку

Також консольний інтерфейс відображає час окремо для перепроектування схеми даних та окремо для перенесення даних (рисунок 12).

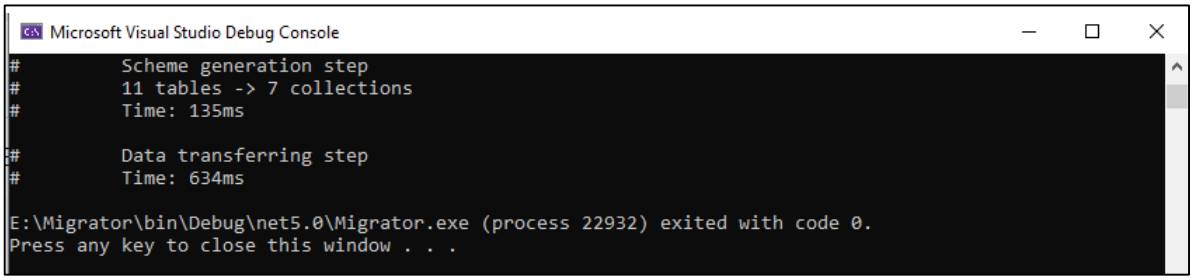

Рисунок 12 – Консольний інтерфейс застосунку

Згенерована документо-орієнтовна схема виводиться у текстовий файл формату txt. Приклад результату зображено на рисунку 13.

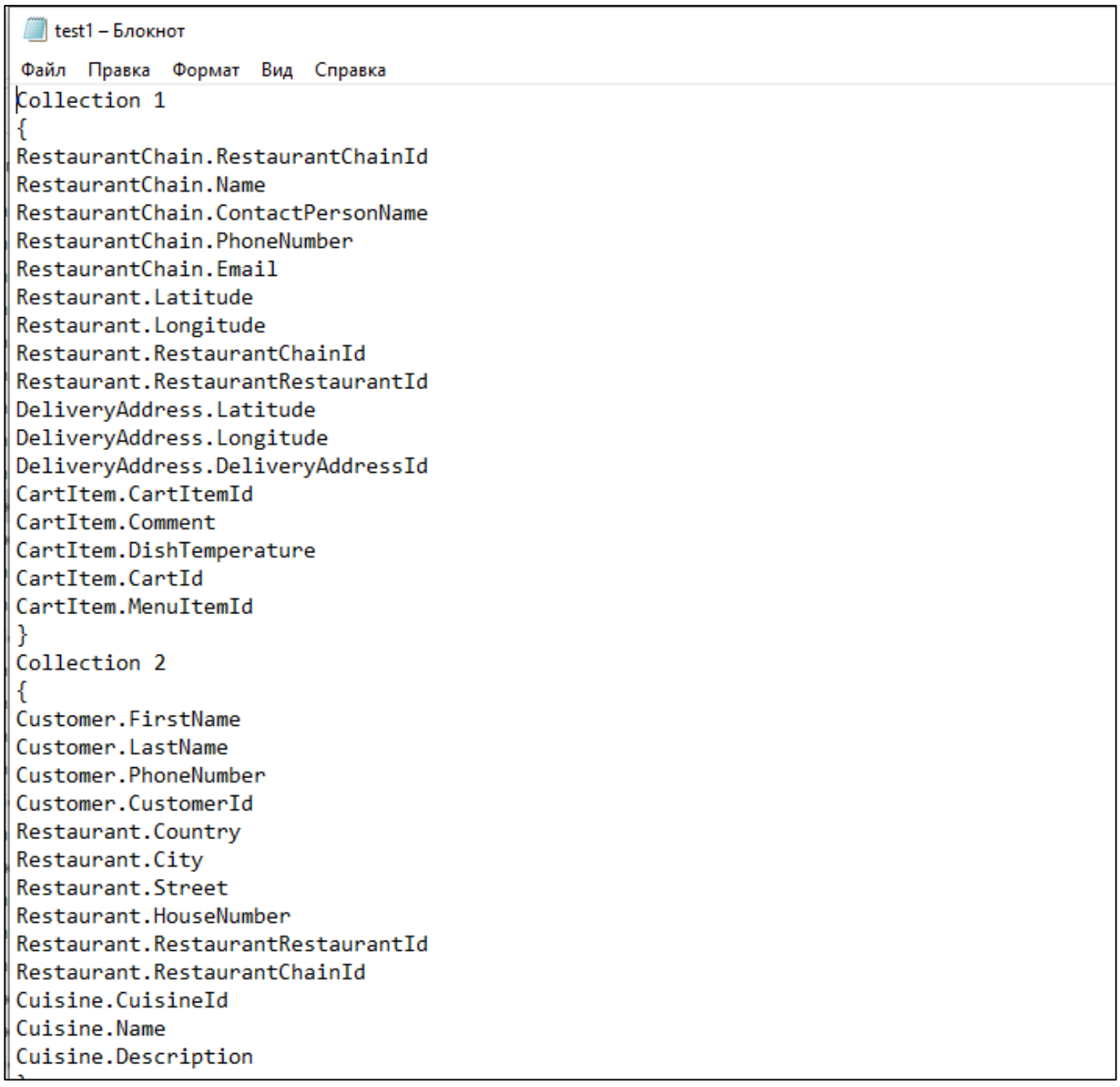

Рисунок 13 – Результуюча схема даних у текстовому файлі

В подальших удосконаленнях можливо додати десктопний або веб-інтерфейс для даного застосунку.

5 ОПИС ЕКСПЕРИМЕНТАЛЬНОГО ДОСЛІДЖЕННЯ

5.1 Проектування реляційної схеми даних

В якості предметної області було обрано сферу харчування, базу даних спроектовано для сайту-агрегатору ресторанів з можливістю зробити онлайн-замовлення.

Після аналізу обраної предметної області було спроектовано реляційну схему бази даних (рисунок 14), ця схема включає в себе такі сутності:

– сутність «Customer» – користувач онлайн-ресурсу, який може зробити замовлення в ресторані;

– сутність «Cart» – електронний кошик користувача;

– сутність «CartItem» – елементи кошика, тобто страви в замовленні;

– сутність «DeliveryAddress» – адреса доставлення замовлення;

– сутність «RestaurantChain» – мережа ресторанів, кафе;

– сутність «Restaurant» – ресторани, кафе;

– сутність «Cuisine» – кухня (українська, японська, італійська тощо);

– сутність «Product» – продукт;

– сутність «Dish» – страва;

– сутність «Ingredient» – інгредієнт, продукт, що міститься у певній страві;

– сутність «MenuItem» – страва – елемент меню ресторану.

Зв’язки між сутностями представлені наступним чином:

– одному користувачеві належить один кошик, тому сутності «Customer» та «Cart» поєднані зв’язком «один-до-одного»;

– в кошику може бути багато елементів замовлення, тому сутність «Cart» поєднана зв’язком «один-до-багатьох» («1 - Б») з сутністю «CartItem»;

– на одну адресу може здійснюватися багато замовлень, тому сутність «DeliveryAddress» поєднана зв’язком «1 - Б» з сутністю «Cart»;

– в мережі ресторанів може бути багато ресторанів, тому сутність «RestaurantChain» поєднана зв’язком «1 - Б» з сутністю «Restaurant»;

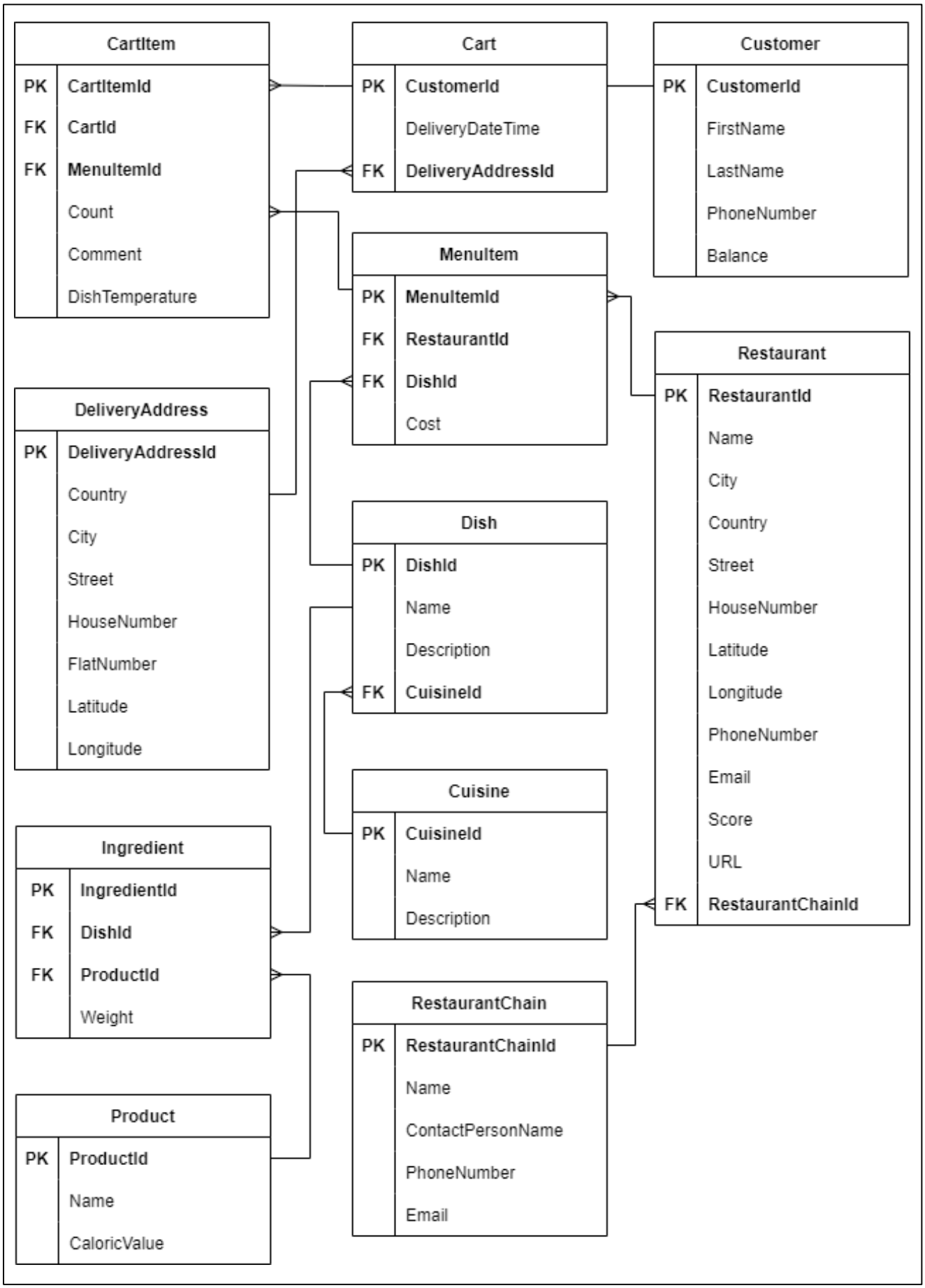

Рисунок 14 – Схема даних для реляційної моделі даних

– до однієї кухні світу може належати багато страв, тому сутність «Cuisine» поєднана зв’язком «1 - Б» з сутністю «Dish»;

– в одному ресторані може пропонуватися багато страв, в той самий час, одна страва може бути запропонована в багатьох ресторанах, тому між цими сутностями побудовано зв’язок «багато-до багатьох» через транзитивну сутність «MenuItem»;

– один продукт може бути використаним для приготування багатьох страв, в той самий час, одна страва може бути приготована за багатьох продуктів, тому між цими сутностями побудовано зв’язок «багато-до багатьох» через транзитивну сутність «Ingredient».

5.2 Створення бази даних MS SQL

На основі спроектованої схеми даних необхідно створити її фізичну модель в СКБД MS SQL за допомогою T-SQL скриптів. Скрипти для створення таблиць наведено в додатку Д.

Після того, як схема даних створена на сервері СКБД, необхідно створити можливість заповнення бази даних випадковими даним (необхідно для того, щоб в подальшому швидко додавати велику кількість даних для експериментів). Для такого додавання були розроблені процедури.

Процедура для додавання даних в таблицю «Customer» виглядає наступним чином (вхідний параметр – це кількість записів для додавання в таблицю, для поля «Balance» в таблиці «Customer» у процедурі задекларовано верхній та нижній ліміти):

```
CREATE PROCEDURE InsertRandomDataIntoCustomerTable(
    @Count INT
)
AS
BEGIN
```

```
    SET NOCOUNT ON

    DECLARE @LowerLimitForBalance DECIMAL(8,3) = 0
    DECLARE @UpperLimitForBalance DECIMAL(8,3) = 10000
    DECLARE @Id INT = 1

    WHILE @Id <= @Count
    BEGIN
        DECLARE @TextId NVARCHAR(10) = CAST(@Id AS NVARCHAR(10));
        INSERT INTO dbo.Customer VALUES (
            'FirstName - ' + @TextId,
            'LastName - ' + @TextId,
            'PhoneNumber - ' + @TextId,
            (@UpperLimitForBalance - @LowerLimitForBalance) * RAND() +
@LowerLimitForBalance
        )
        SET @Id = @Id + 1
    END
END
```

Для заповнення таблиць із зовнішніми ключами необхідно у якості вхідних параметрів передавати верхній та нижній ліміти для цих полів.

Розглянемо таблицю «MenuItem», ця таблиця містить два зовнішні ключі «RestaurantId» та «DishId». Це означає, що, по-перше, таблиці «Restaurant» та «Dish» мають бути заповнені даними раніше, ніж таблиця «MenuItem», та, по-друге в якості вхідних параметрів до процедури повинні бути додані ліміти для «Id» відповідно до значень полів «RestaurantId» та «DishId». Процедура для заповнення таблиці «MenuItem» виглядає наступним чином:

```
CREATE PROCEDURE InsertRandomDataIntoMenuItemTable(
    @Count INT,
    @LowerLimitForRestaurantId INT,
    @UpperLimitForRestaurantId INT,
    @LowerLimitForDishId INT,
    @UpperLimitForDishId INT
)
AS
BEGIN
    SET NOCOUNT ON

    DECLARE @LowerLimitForCost DECIMAL(8,3) = 10
    DECLARE @UpperLimitForCost DECIMAL(8,3) = 5000
    DECLARE @Id INT = 1

    WHILE @Id <= @Count
    BEGIN

        DECLARE @TextId NVARCHAR(10) = CAST(@Id AS NVARCHAR(10));
```

```
        INSERT INTO dbo.MenuItem VALUES (
            ROUND(((@UpperLimitForRestaurantId -
@LowerLimitForRestaurantId) * Rand()) + @LowerLimitForRestaurantId, 0),
            ROUND(((@UpperLimitForDishId - @LowerLimitForDishId) * Rand())
+ @LowerLimitForDishId, 0),
            (@UpperLimitForDishId - @LowerLimitForCost) * Rand() +
@LowerLimitForCost
        )

        SET @Id = @Id + 1
    END
END
```

Аналогічним чином було створено процедури для додавання випадкових даних до всіх інших таблиць.

5.3 Розробка запитів до бази даних

Проаналізувавши предметну область на створену на попередньому кроці схему бази даних будо складено тестові запити до цієї бази даних, що вірогідно можуть бути корисними у програмній системі, яка використовує таку схему даних.

Запит, що дозволяє отримати список страв (назву та опис) з наборами продуктів-інгредієнтів має такий вигляд:

```
SELECT Dish.Name, Dish.Description,
    (SELECT STRING_AGG(Product.Name, ' ')
        FROM Product JOIN Ingredient ON Product.ProductId =
Ingredient.ProductId
        WHERE Ingredient.DishId = DishId)
    FROM Dish GROUP BY Dish.Name, Dish.Description
```

Запит, що дозволяє отримати список продуктів з їхньою калорійністю (за замовчуванням), що є інгредієнтами певної страви, має такий вигляд:

```
SELECT Product.Name, Product.CaloricValue
    FROM Product JOIN Ingredient ON Product.ProductId =
Ingredient.ProductId
    WHERE Ingredient.DishId = 3
```

Запит, що дозволяє отримати список продуктів з фактичною калорійністю (з урахуванням ваги), що є інгредієнтами певної страви, має такий вигляд:

```
SELECT Product.Name, SUM(Product.CaloricValue * Ingredient.Weight)
    FROM Product JOIN Ingredient ON Product.ProductId =
Ingredient.ProductId
    WHERE Ingredient.DishId = 3
    GROUP BY Product.Name
```

Запит, що дозволяє отримати список продуктів з фактичною калорійністю (з урахуванням ваги), що є інгредієнтами страв, має такий вигляд:

```
SELECT Product.Name, SUM(Product.CaloricValue * Ingredient.Weight)
    FROM Product JOIN Ingredient ON Product.ProductId =
Ingredient.ProductId
    GROUP BY Product.Name
```

Запит, що дозволяє отримати максимальну калорійність страви, має такий вигляд:

```
SELECT MAX(a.TotalCaloricValue) AS 'TotalCaloricValue'
    FROM (SELECT Dish.Name, SUM(Product.CaloricValue * Ingredient.Weight)
AS TotalCaloricValue
        FROM Dish JOIN Ingredient ON Dish.DishId = Ingredient.DishId
            JOIN Product ON Product.ProductId = Ingredient.ProductId
        GROUP BY Dish.Name) AS A
```

Запит, що дозволяє отримати список страв з найменшою калорійністю у своїй групі (за назвою), має такий вигляд:

```
SELECT A.Name, MIN(a.TotalCaloricValue) AS 'TotalCaloricValue'
    FROM (SELECT Dish.Name, SUM(Product.CaloricValue * Ingredient.Weight)
AS TotalCaloricValue
        FROM Dish JOIN Ingredient ON Dish.DishId = Ingredient.DishId
            JOIN Product ON Product.ProductId = Ingredient.ProductId
        GROUP BY Dish.Name)
    AS A GROUP BY A.Name
```

Запит, що дозволяє отримати список страв з найбільшою калорійністю, має такий вигляд:

```
SELECT Dish.Name, SUM(Product.CaloricValue * Ingredient.Weight) AS
TotalCaloricValue
    FROM Dish JOIN Ingredient ON Dish.DishId = Ingredient.DishId
        JOIN Product ON Product.ProductId = Ingredient.ProductId GROUP BY
Dish.Name
        HAVING SUM(Product.CaloricValue * Ingredient.Weight) = (SELECT TOP
1 SUM(Product.CaloricValue * Ingredient.Weight)
            FROM Dish JOIN Ingredient ON Dish.DishId = Ingredient.DishId
                JOIN Product ON Product.ProductId = Ingredient.ProductId
GROUP BY Dish.Name ORDER BY 1 DESC)
```

Запит, що дозволяє отримати список страв з їх цінами для певного ресторану, має такий вигляд:

```
SELECT Dish.Name, Dish.Description, MenuItem.Cost
    FROM Dish JOIN MenuItem ON Dish.DishId = MenuItem.DishId
        WHERE MenuItem.RestaurantId = 1
```

Запит, що дозволяє отримати список страв з їх цінами для ресторанів, назва яких підпадає під умову пошуку (за частиною слова), має такий вигляд:

```
SELECT Dish.DishId, Dish.Name, Dish.Description, MenuItem.Cost
    FROM Dish JOIN MenuItem ON Dish.DishId = MenuItem.DishId
        JOIN Restaurant ON Restaurant.RestaurantId = MenuItem.RestaurantId
        WHERE Restaurant.Name LIKE '%'
```

Запит, що дозволяє отримати інформацію про клієнтів, які мають нульовий баланс, має такий вигляд:

```
SELECT Customer.CustomerId, Customer.FirstName, Customer.LastName,
Customer.PhoneNumber, Customer.Balance
    FROM Customer
    WHERE Balance = 0
```

Запит, що дозволяє отримати список у вигляді: продукт, страва, до якої він належить, та вага цього продукту у страві, де назва продукта відповідає певному значенню, має такий вигляд:

```
SELECT Product.Name, Dish.Name, Ingredient.Weight
    FROM Ingredient
        JOIN Dish ON Ingredient.DishId = Dish.DishId
```

```
        JOIN Product ON Ingredient.ProductId = Product.ProductId
        WHERE Product.Name = ''
```

Запит, що дозволяє отримати інформацію по певному кошику із зазначенням загальної вартості товарів у цьому кошику, має такий вигляд:

```
SELECT Cart.CustomerId, Cart.DeliveryAddressId, SUM(CartItem.Count *
MenuItem.Cost) AS 'TotalCost'
    FROM CartItem JOIN MenuItem ON CartItem.MenuItemId =
MenuItem.MenuItemId
        JOIN Cart ON CartItem.CartId = Cart.CustomerId
        JOIN Dish ON Dish.DishId = MenuItem.DishId
        WHERE Cart.CustomerId = 1
    GROUP BY Cart.CustomerId, Cart.DeliveryAddressId
```

Запит, що дозволяє отримати інформацію по певному кошику із зазначенням страв та кількості одиниць кожної страви, з визначеною датою доставки та визначеною адресою доставки, має такий вигляд:

```
SELECT CartItem.Count, Cart.DeliveryAddressId, Dish.Name, CartItem.Count *
MenuItem.Cost AS 'DishesCost'
    FROM CartItem JOIN MenuItem ON CartItem.MenuItemId =
MenuItem.MenuItemId
        JOIN Cart ON CartItem.CartId = Cart.CustomerId
        JOIN Dish ON Dish.DishId = MenuItem.DishId
        WHERE Cart.CustomerId = 1
            AND Cart.DeliveryDateTime = '10/10/2022'
            AND Cart.DeliveryAddressId = 2
```

Запит, що дозволяє отримати інформацію по кошикам, загальна вартість товарів в яких менша, ніж баланс користувача, має такий вигляд:

```
SELECT Cart.CustomerId, SUM(CartItem.Count * MenuItem.Cost) AS 'TotalCost'
    FROM CartItem JOIN MenuItem ON CartItem.MenuItemId =
MenuItem.MenuItemId
        JOIN Cart ON CartItem.CartId = Cart.CustomerId
    JOIN Customer ON Cart.CustomerId = Customer.CustomerId
    GROUP BY Cart.CustomerId
    HAVING 'TotalCost' < Customer.Balance
```

Запит, що дозволяє отримати інформацію вартості товарів в кошиках (з урахуванням кількості товарів), з заданою датою доставки, має такий вигляд:

```
SELECT SUM(CartItem.Count * MenuItem.Cost)
    FROM CartItem
        JOIN MenuItem ON CartItem.MenuItemId = MenuItem.MenuItemId
        JOIN Cart ON CartItem.CartId = Cart.CustomerId
        WHERE Cart.DeliveryDateTime = ''
```

Запит, що дозволяє отримати списки страв із зазначенням кухні, до якої належить кожна страва, має такий вигляд:

```
SELECT Dish.Name, Dish.Description, Cuisine.Name, Cuisine.Description
    FROM Cuisine JOIN Dish ON Cuisine.CuisineId = Dish.CuisineId
```

Запит, що дозволяє отримати інформацію по рестораном з певною адресою, має такий вигляд:

```
SELECT Restaurant.RestaurantId, Restaurant.Name, Restaurant.PhoneNumber,
Restaurant.Email, Restaurant.Score, Restaurant.Url
    FROM Restaurant
        WHERE Restaurant.City = ''
            AND Restaurant.Country = ''
            AND Restaurant.Street = ''
            AND Restaurant.HouseNumber = ''
```

Запит, що дозволяє отримати інформацію по адресам за певною вулицею, номером будинку та номером квартири, має такий вигляд:

```
SELECT DeliveryAddress.DeliveryAddressId, DeliveryAddress.Country,
DeliveryAddress.City
    FROM DeliveryAddress
        WHERE DeliveryAddress.Street = '' AND DeliveryAddress.HouseNumber =
'' AND DeliveryAddress.FlatNumber = 10
```

Запит, що дозволяє отримати адресу доставки за певним кошиком, має такий вигляд:

```
SELECT DeliveryAddress.DeliveryAddressId, DeliveryAddress.Country,
DeliveryAddress.City, DeliveryAddress.Street,
  DeliveryAddress.HouseNumber, DeliveryAddress.FlatNumber
    FROM DeliveryAddress
    JOIN Cart ON DeliveryAddress.DeliveryAddressId = Cart.
DeliveryAddressId
        WHERE Cart.CustomerId = 2
```

Запит, що дозволяє інформацію по ресторану за адресою сайту ресторану, має такий вигляд:

```
SELECT Restaurant.RestaurantId, Restaurant.Name, Restaurant.PhoneNumber,
Restaurant.Email, Restaurant.Score
    FROM Restaurant
       WHERE Restaurant.Url = ''
```

Вищенаведені запити задовольнятимуть потреби користувачів програмної системи зі схемою даних, розробленою в рамках даної роботи.

5.4 Проведення дослідження

Дослідження включало в себе перепроектування бази даних за розробленим алгоритмом, виконання міграції даних на отриману схему, та на дві інші схеми, отримані за допомогою існуючих методів міграції, і виконання запитів у трьох отриманих базах даних.

Експериментальне дослідження проводилося на персональному комп’ютері з процесором Intel(R) Core(TM) i5-8400U CPU @ 2.80GHz з операційною системою Windows 10, MS SQL Server 2019 Express версії 15.0.2, MongoDB Community Server версії 4.2.0.

З реляційної схеми даних, що містить 11 таблиць, в результаті перепроектування під документо-орієнтовний стиль було отримано схему даних на 7 колекцій.

Схема даних після перепроектування виглядає наступним чином:

```
Collection 1
{
      RestaurantChain.RestaurantChainId
      RestaurantChain.Name
      RestaurantChain.ContactPersonName
      RestaurantChain.PhoneNumber
```

```
      RestaurantChain.Email
      Restaurant.Latitude
      Restaurant.Longitude
      Restaurant.RestaurantChainId
      Restaurant.RestaurantRestaurantId
      DeliveryAddress.Latitude
      DeliveryAddress.Longitude
      DeliveryAddress.DeliveryAddressId
      CartItem.CartItemId
      CartItem.Comment
      CartItem.DishTemperature
      CartItem.CartId
      CartItem.MenuItemId
}

Collection 2
{
      Customer.FirstName
      Customer.LastName
      Customer.PhoneNumber
      Customer.CustomerId
      Restaurant.Country
      Restaurant.City
      Restaurant.Street
      Restaurant.HouseNumber
      Restaurant.RestaurantRestaurantId
      Restaurant.RestaurantChainId
      Cuisine.CuisineId
      Cuisine.Name
      Cuisine.Description

}

Collection 3
{
      Customer.Balance
      Customer.CustomerId
      DeliveryAddress.City
      DeliveryAddress.Country
      DeliveryAddress.DeliveryAddressId
      DeliveryAddress.FlatNumber
      DeliveryAddress.HouseNumber
      DeliveryAddress.Street
}

Collection 4
{
      Restaurant.Email
      Restaurant.Name
      Restaurant.PhoneNumber
      Restaurant.RestaurantChainId
      Restaurant.RestaurantRestaurantId
      Restaurant.Score
      Restaurant.URL

}
```

```
Collection 5
{
      Cart.CustomerId
      Cart.DeliveryAddressId
      Cart.DeliveryDateTime
      CartItem.CartId
      CartItem.CartItemId
      CartItem.Count
      CartItem.MenuItemId
      MenuItem.Cost
      MenuItem.DishId
      MenuItem.MenuItemId
      MenuItem.RestaurantId
}

Collection 6
{
      Ingredient.DishId
      Ingredient.IngredientId
      Ingredient.ProductId
      Ingredient.Weigth
      Product.CaloricValue
      Product.Name
      Product.ProductId
}

Collection 7
{
      Dish.CuisineId
      Dish.Description
      Dish.DishId
      Dish.Name
}
```

Генерацію документо-орієнтовної схеми було запущено 5 разів задля визначення середнього часу міграції схеми даних, час роботи перепроектування становив у середньому 150 мс.

Після перепроектування схеми даних було проведено міграцію даних до MongoDB. Окрім отриманої в результаті алгоритму схеми даних, для порівняння було взято до розгляду схему, де колекції повністю відповідають таблицям з реляційної схеми, та схему, де усі таблиці поміщено в одну колекцію. Тестування міграції проводилося на різних об’ємах даних, а саме: 10000, 20000, 50000, 100000, 300000, 600000, 1000000 елементів.

В таблиці 1 наведено час виконання міграції при різних кількостях елементів для усіх трьох алгоритмів (час роботи замірявся в розробленому програмному застосунку).

Таблиця 1 – Час міграції даних для різної кількості елементів

Кількість елементів у таблицях	Час міграції, мс		
	Міграція схеми «1 таблиця -> 1 колекція»	Міграція схеми «усі таблиці -> 1 колекція»	Міграція схеми за розробленим алгоритмом
10000	1527	1402	2131
20000	2562	1912	3102
50000	6980	5741	8527
100000	11687	9054	13256
300000	37219	32854	48411
600000	62046	50573	70738
1000000	114777	95286	149856

Як видно з вищенаведеної таблиці, найшвидше міграція пройшла з варіантом схеми даних, де усі таблиці з реляційної схеми об’єднуються в одну колекцію документо-орієнтовної схеми. Середні показники серед трьох схем показала міграція, де кожній реляційній таблиці відповідає аналогічна документо-орієнтовна колекція. Найбільшим виявився час міграції до схеми, розробленої в рамках даної роботи – алгоритмом на основі запитів. Такі результати є очікуваними та пояснюються наступним чином:

– варіант зі схемою «усі таблиці -> одна колекція» є найшвидшим через те, що кількість запитів на додавання даних в MongoDB істотно менша, аніж при інших варіантах схеми, адже замість того, щоб додавати документи у n колекцій, дані додаються в одну колекцію;

– варіант зі схемою «одна таблиця -> одна колекція» виявився довшим, ніж при об’єднанні в одну колекцію через те, що запитів на додавання документів було стільки ж, скільки даних містилося в кожній колекції, тобто, порівняно з попереднім методом, час додатково було витрачено на роботу з MongoDB;

– варіант зі схемою, що була згенерована за алгоритмом на основі запитів є найдовшим через те, що при вибірці даних з MS SQL потрібно було виконати більше запитів, щоб зібрати дані для однієї колекції, тобто, якщо для колекції необхідні дані з трьох таблиць, то відповідно, для формування одного документу, до реляційної бази було виконано три запити, кожен з яких дістає дані з різної таблиці.

На рисунку 15 зображено графік залежності часу міграції даних від кількості елементів в таблицях бази даних для кожної схеми даних, що розглядається.

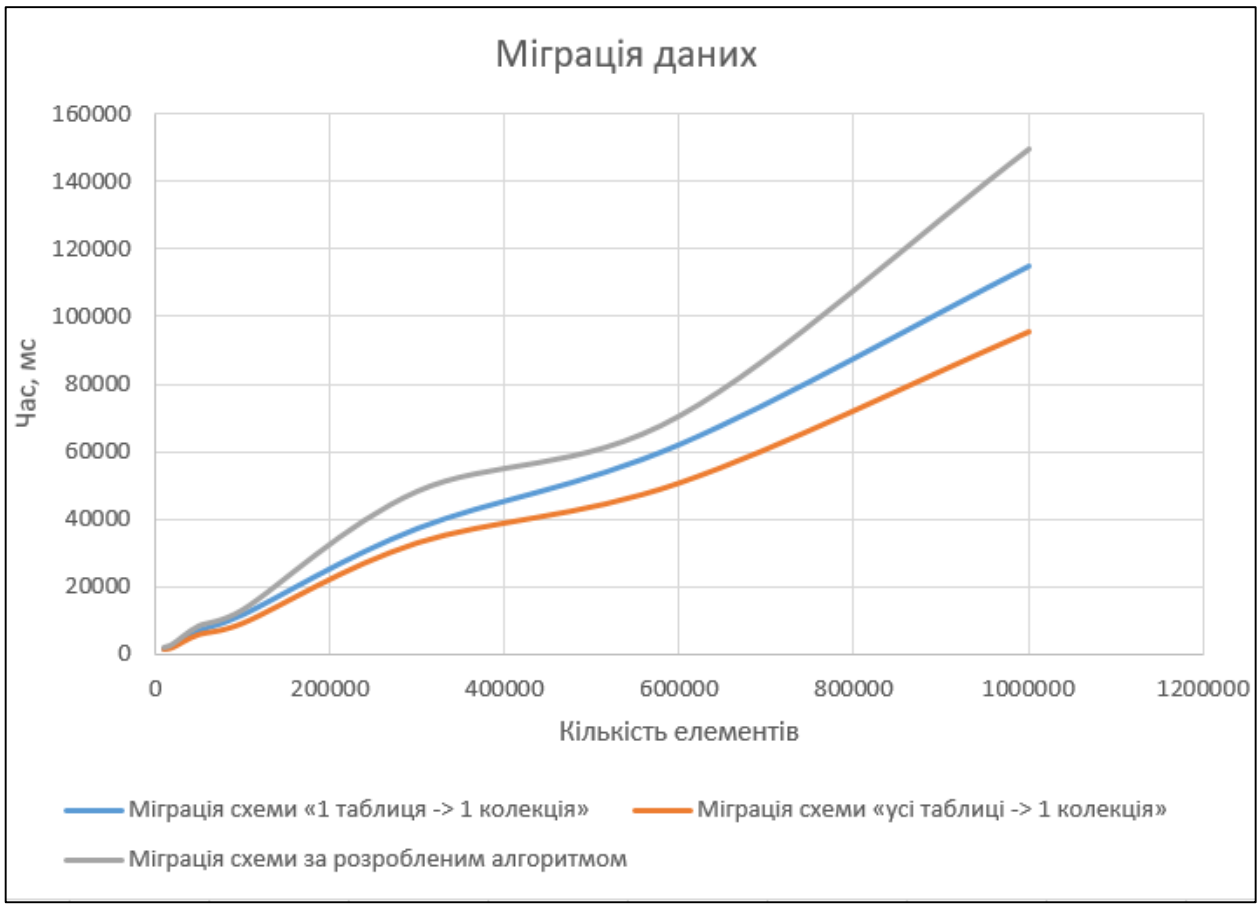

Рисунок 15 – Залежність часу міграції від кількості елементів для різних схем даних

Час міграції не є основним показником успішності алгоритму підтримки міграції схеми даних, оскільки міграція – це одноразовий процес, обмежений у часі, більш вагомим є показник швидкості виконання запитів у цільовій базі, адже

запити виконуються у системі постійно, та саме час їх виконання і впливає на швидкість функціонування програмних систем, які користуються цією базою даних.

Для тестування ефективності отриманої в результаті роботи алгоритму схеми даних запити до бази даних (список запитів наведено в попередньому розділі) було адаптовано для роботи в MongoDB та запущено для різних варіантів документо-орієнтовної схеми даних.

Тестування запитів на цільових базах даних відбувалося у Robo 3T – це менеджер підключень MongoDB, який має зручний графічний інтерфейс для MongoDB, кросплатформену оболонку та підтримується JSON, його оболонка може бути вбудована в Mongo Shell з доступом як у Mongo CLI, так і в Mongo GUI. Robo 3T — це програмне забезпечення, яке використовує невелику кількість ресурсів, доступних на машині, тобто є високопродуктивним рішенням. За допомогою Robo 3T користувач може переглядати, редагувати та видаляти документи mongo. Крім того, Robo 3T є волонтерським проектом з відкритим кодом, і він є абсолютно безкоштовним для громадськості.

Зважаючи на переваги та доступність цього програмного забезпечення, саме Robo 3T було обране для проведення досліджень з цільовими базами даних MongoDB.

Кожен запит було запущено по 10 разів, зафіксовано час виконання (Robo 3T відображує час виконання запиту після його відпрацювання) та усереднено час виконання кожного з них задля підвищення об’єктивності дослідження та зменшення похибок вимірювань.

Наведемо ці запити у схематичному представленні (у вигляді множин використовуваних в них полів), а також наведемо результати часу виконання цих запитів.

Query1 = { DishName, DishDescription, ProductName, ProductProductId, IngredientProductId, IngredientDishId, DishDishId }.

В таблиці 2 відображено час виконання першого запиту для різної кількості даних для різних методів міграції схеми даних.

Таблиця 2 – Час виконання першого запиту

Кількість елементів у таблицях	Час виконання запиту, мс		
	Міграція схеми «1 таблиця -> 1 колекція»	Міграція схеми «усі таблиці -> 1 колекція»	Міграція схеми за розробленим алгоритмом
10000	78	50	65
20000	149	94	102
50000	214	337	153
100000	769	1463	698
300000	3745	4739	2137

Рисунок 16 відображає залежність часу виконання першого запиту від кількості елементів у таблицях для схем даних, побудованих за різними алгоритмами.

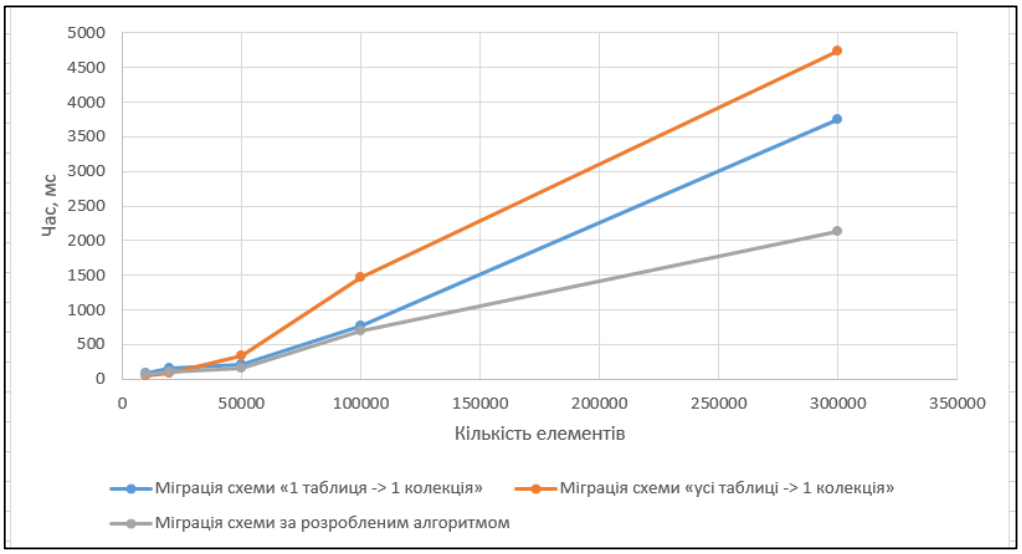

Рисунок 16 – Залежність часу виконання першого запиту від методу міграції схеми та кількості елементів

Query2 = { ProductName, ProductCaloricValue, ProductProductId, IngredientProductId, IngredientDishId }. В таблиці 3 відображено час виконання другого запиту для різної кількості даних для різних методів міграції схеми даних.

Таблиця 3 – Час виконання другого запиту

Кількість елементів у таблицях	Час виконання запиту, мс		
	Міграція схеми «1 таблиця -> 1 колекція»	Міграція схеми «усі таблиці -> 1 колекція»	Міграція схеми за розробленим алгоритмом
10000	58	39	32
20000	129	129	74
50000	274	253	121
100000	409	373	234
300000	835	756	683

Рисунок 17 відображає залежність часу виконання другого запиту від кількості елементів для різних алгоритмів.

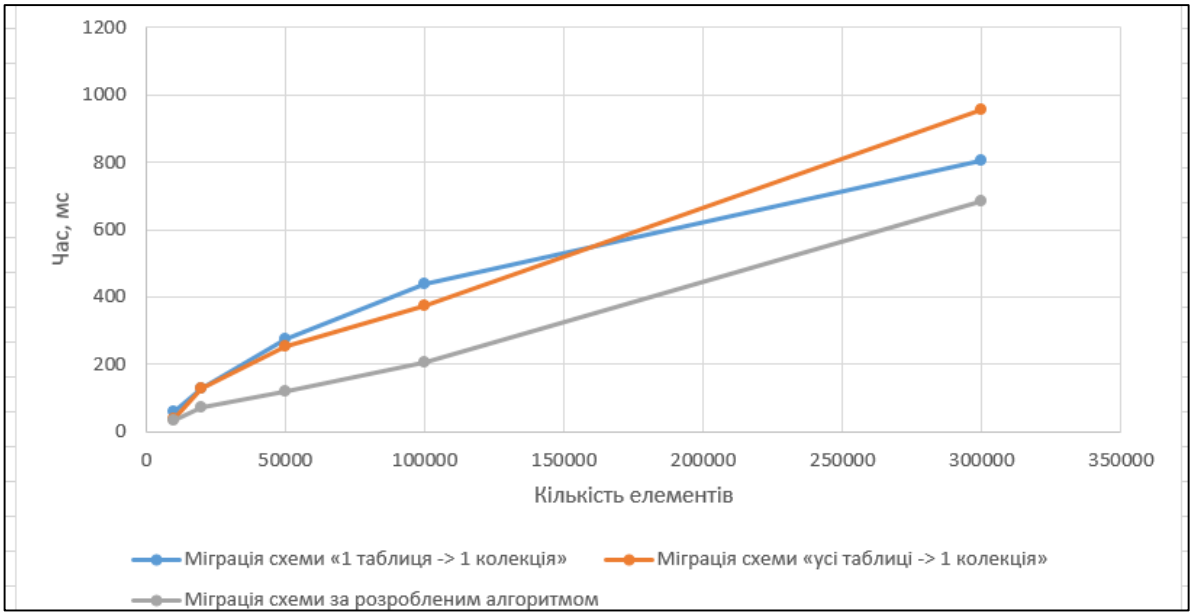

Рисунок 17 – Залежність часу виконання другого запиту від методу міграції схеми та кількості елементів

Query3 = { ProductName, ProductCaloricValue, IngredientWeigth, ProductProductId, IngredientProductId, IngredientDishId }. В таблиці 4 відображено час виконання третього запиту для різної кількості даних.

Таблиця 4 – Час виконання третього запиту

Кількість елементів у таблицях	Час виконання запиту, мс		
	Міграція схеми «1 таблиця -> 1 колекція»	Міграція схеми «усі таблиці -> 1 колекція»	Міграція схеми за розробленим алгоритмом
10000	64	45	60
20000	159	174	140
50000	315	383	274
100000	509	592	412
300000	1683	1854	1245

Рисунок 18 відображає залежність часу виконання третього запиту від кількості елементів для різних алгоритмів.

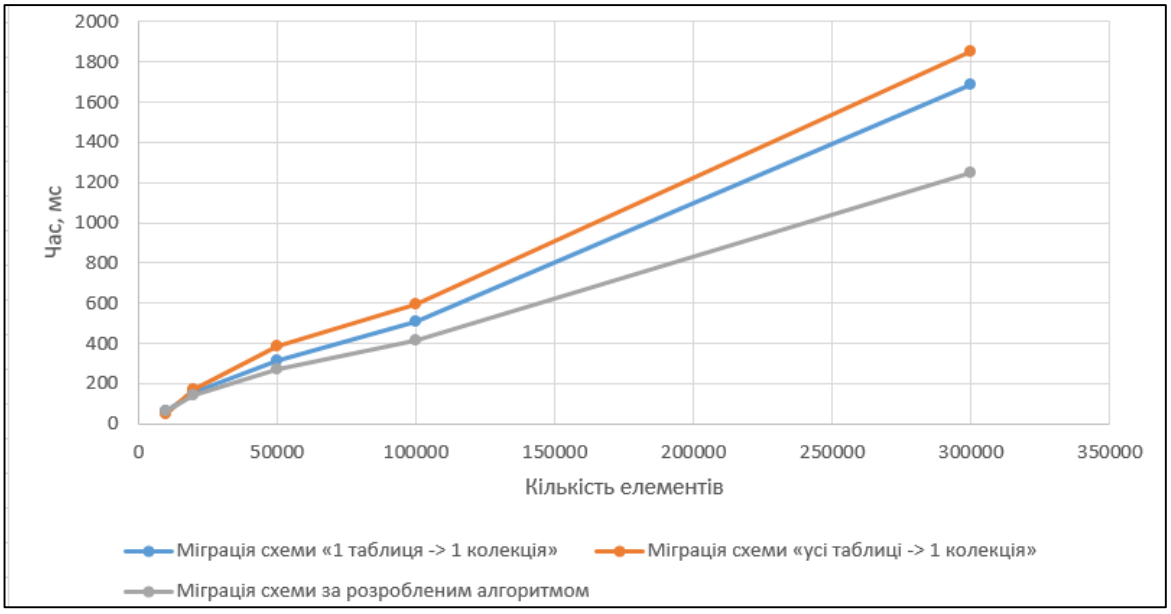

Рисунок 18 – Залежність часу виконання третього запиту від методу міграції схеми та кількості елементів

Query4 = { ProductName, ProductCaloricValue, IngredientWeigth, ProductProductId, IngredientProductId }. В таблиці 5 відображено час виконання четвертого запиту для різної кількості даних для різних методів міграції схеми.

Таблиця 5 – Час виконання четвертого запиту

Кількість елементів у таблицях	Час виконання запиту, мс		
	Міграція схеми «1 таблиця -> 1 колекція»	Міграція схеми «усі таблиці -> 1 колекція»	Міграція схеми за розробленим алгоритмом
10000	70	53	49
20000	163	140	165
50000	459	348	382
100000	694	733	613
300000	2182	2642	1724

Рисунок 19 відображає залежність часу виконання четвертого запиту від кількості елементів для різних алгоритмів.

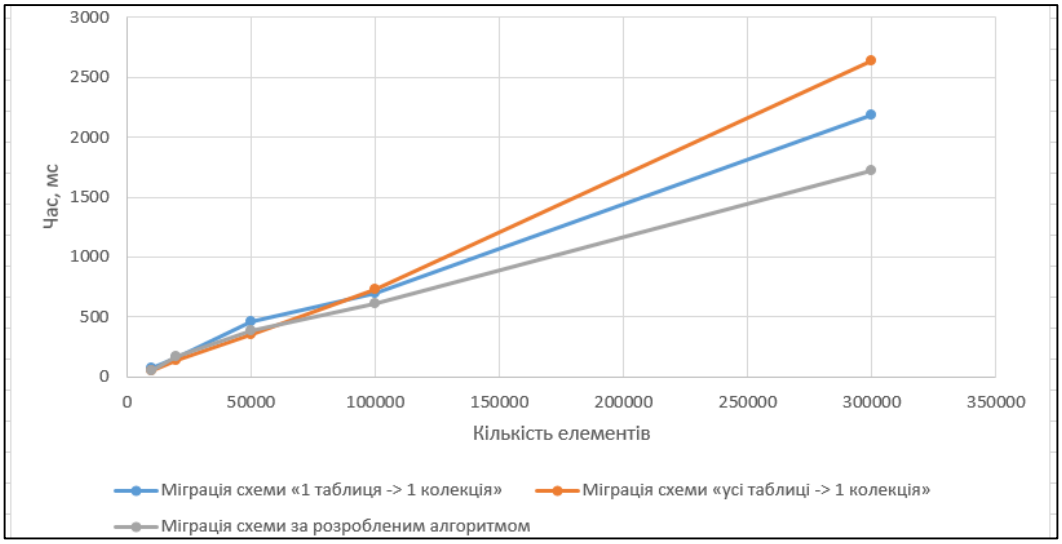

Рисунок 19 – Залежність часу виконання четвертого запиту від методу міграції схеми та кількості елементів

Query5 = { ProductCaloricValue, IngredientWeigth, DishName, DishDishId, IngredientDishId, ProductProductId, IngredientProductId }. В таблиці 6 відображено час виконання п’ятого запиту для різної кількості даних для різних методів міграції схеми даних.

Таблиця 6 – Час виконання п’ятого запиту

Кількість елементів у таблицях	Час виконання запиту, мс		
	Міграція схеми «1 таблиця -> 1 колекція»	Міграція схеми «усі таблиці -> 1 колекція»	Міграція схеми за розробленим алгоритмом
10000	175	139	164
20000	279	245	250
50000	683	764	518
100000	1385	1593	1043
300000	2935	4234	2863

Рисунок 20 відображає залежність часу виконання п’ятого запиту від кількості елементів для різних алгоритмів.

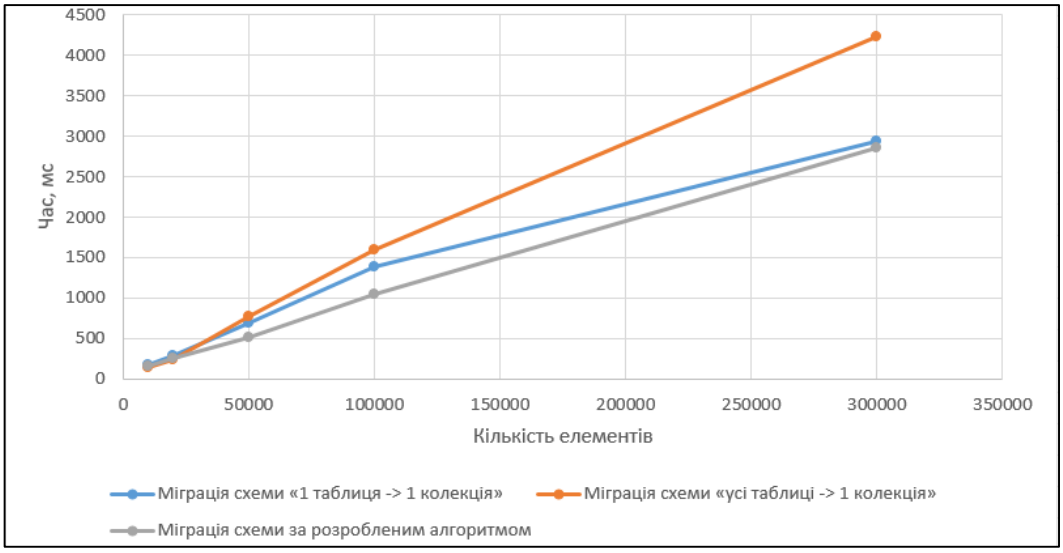

Рисунок 20 – Залежність часу виконання п’ятого запиту від методу міграції схеми та кількості елементів

Query6 = { DishName, ProductCaloricValue, IngredientWeigth, DishDishId, IngredientDishId, ProductProductId, IngredientProductId }. В таблиці 7 відображено час виконання шостого запиту для різної кількості даних для різних методів міграції схеми даних.

Таблиця 7 – Час виконання шостого запиту

Кількість елементів у таблицях	Час виконання запиту, мс		
	Міграція схеми «1 таблиця -> 1 колекція»	Міграція схеми «усі таблиці -> 1 колекція»	Міграція схеми за розробленим алгоритмом
10000	182.35	141	160
20000	268	243	259
50000	671	793	530
100000	1483	1705	1242
300000	3863	3996	2593

Рисунок 21 відображає залежність часу виконання шостого запиту від кількості елементів для різних алгоритмів.

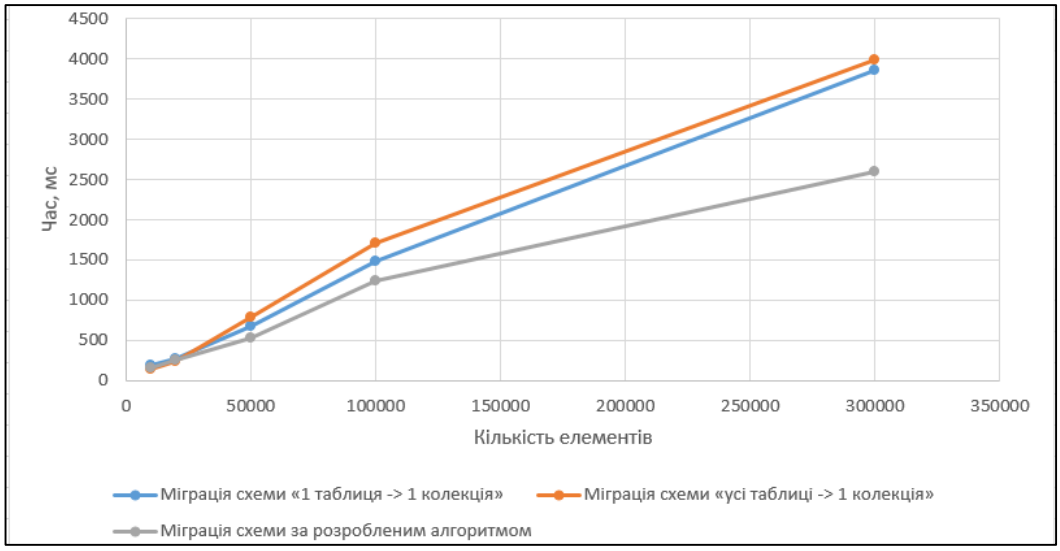

Рисунок 21 – Залежність часу виконання шостого запиту від методу міграції схеми та кількості елементів

Query7 = { DishName, ProductCaloricValue, IngredientWeigth, DishDishId, IngredientDishId, ProductProductId, IngredientProductId }. В таблиці 8 відображено час виконання сьомого запиту для різної кількості даних для різних методів міграції схеми даних.

Таблиця 8 – Час виконання сьомого запиту

Кількість елементів у таблицях	Час виконання запиту, мс		
	Міграція схеми «1 таблиця -> 1 колекція»	Міграція схеми «усі таблиці -> 1 колекція»	Міграція схеми за розробленим алгоритмом
10000	203	181	194
20000	385	296	397
50000	594	762	553
100000	1493	1838	1130
300000	4920	5989	4149

Рисунок 22 відображає залежність часу виконання сьомого запиту від кількості елементів для різних алгоритмів.

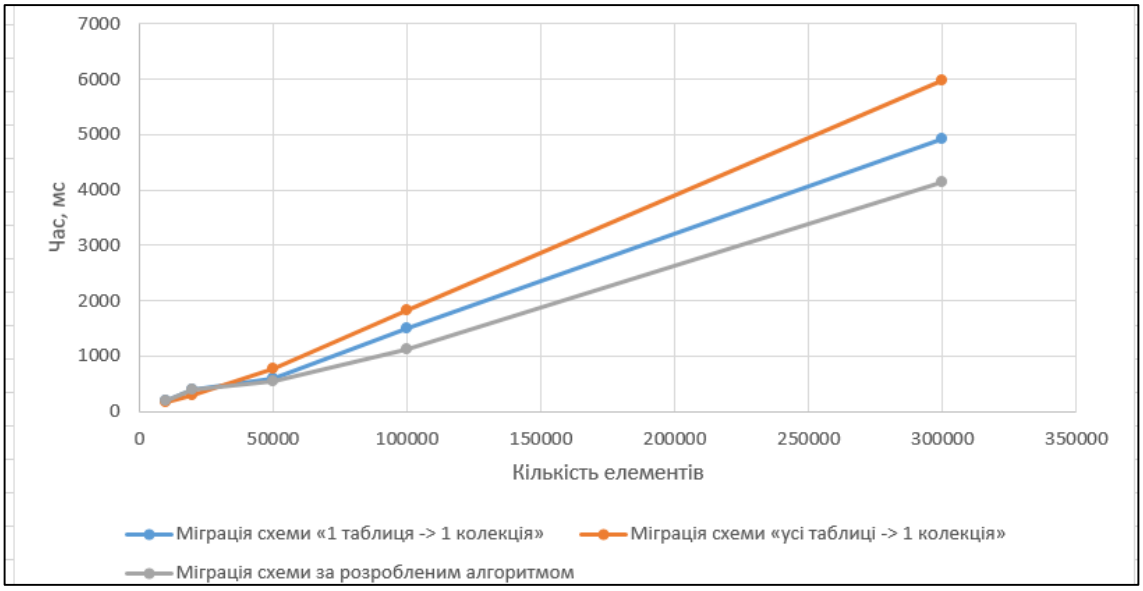

Рисунок 22 – Залежність часу виконання сьомого запиту від методу міграції схеми та кількості елементів

Query8 = { DishName, DishDishId, MenuItemDishId, MenuItemRestaurantId }. В таблиці 9 відображено час виконання восьмого запиту для різної кількості даних для різних методів міграції схеми даних.

Таблиця 9 – Час виконання восьмого запиту

Кількість елементів у таблицях	Час виконання запиту, мс		
	Міграція схеми «1 таблиця -> 1 колекція»	Міграція схеми «усі таблиці -> 1 колекція»	Міграція схеми за розробленим алгоритмом
10000	40	46	31
20000	73	84	67
50000	137	129	135
100000	239	380	202
300000	742	797	598

Рисунок 23 відображає залежність часу виконання восьмого запиту від кількості елементів для різних алгоритмів.

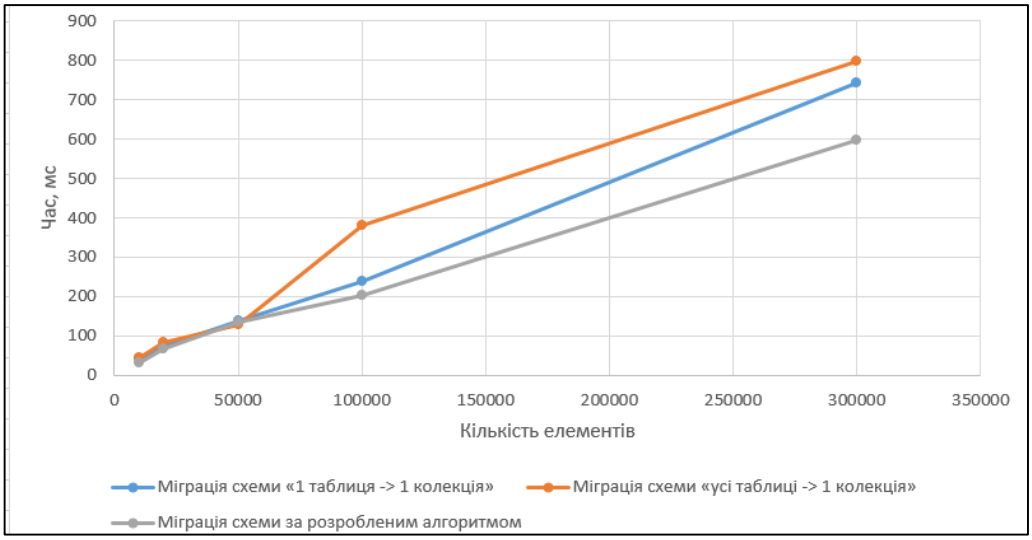

Рисунок 23 – Залежність часу виконання восьмого запиту від методу міграції схеми та кількості елементів

Query9 = { DishName, DishDishId, DishDescription, MenuItemCost, MenuItemDishId, RestaurantRestaurantId, MenuItemRestaurantId, RestaurantName }. В таблиці 10 відображено час виконання дев’ятого запиту для різної кількості даних для різних методів міграції схеми даних.

Таблиця 10 – Час виконання дев’ятого запиту

Кількість елементів у таблицях	Час виконання запиту, мс		
	Міграція схеми «1 таблиця -> 1 колекція»	Міграція схеми «усі таблиці -> 1 колекція»	Міграція схеми за розробленим алгоритмом
10000	62	51	70
20000	113	108	136
50000	243	220	250
100000	406	634	400
300000	1143	2589	1150

Рисунок 24 відображає залежність часу виконання дев’ятого запиту від кількості елементів для різних алгоритмів.

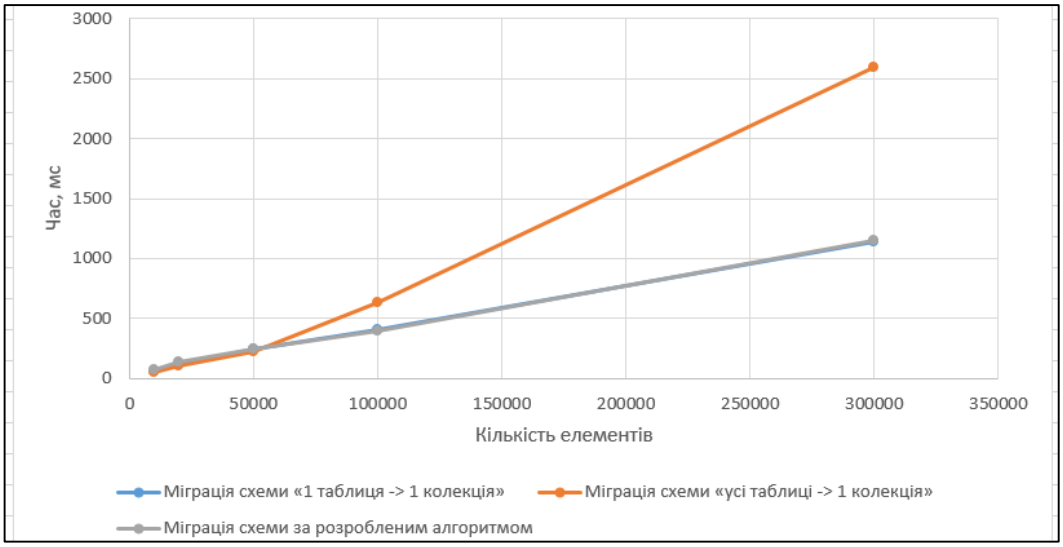

Рисунок 24 – Залежність часу виконання дев’ятого запиту від методу міграції схеми та кількості елементів

Query10 = { CustomerCustomerId, CustomerFirstName, CustomerLastName, CustomerPhoneNumber, CustomerBalance }. В таблиці 11 відображено час виконання десятого запиту для різної кількості даних для різних методів міграції схеми даних.

Таблиця 11 – Час виконання десятого запиту

Кількість елементів у таблицях	Час виконання запиту, мс		
	Міграція схеми «1 таблиця -> 1 колекція»	Міграція схеми «усі таблиці -> 1 колекція»	Міграція схеми за розробленим алгоритмом
10000	30	34	45
20000	98	90	103
50000	204	364	263
100000	356	484	403
300000	895	1259	1046

Рисунок 25 відображає залежність часу виконання десятого запиту від кількості елементів для різних алгоритмів.

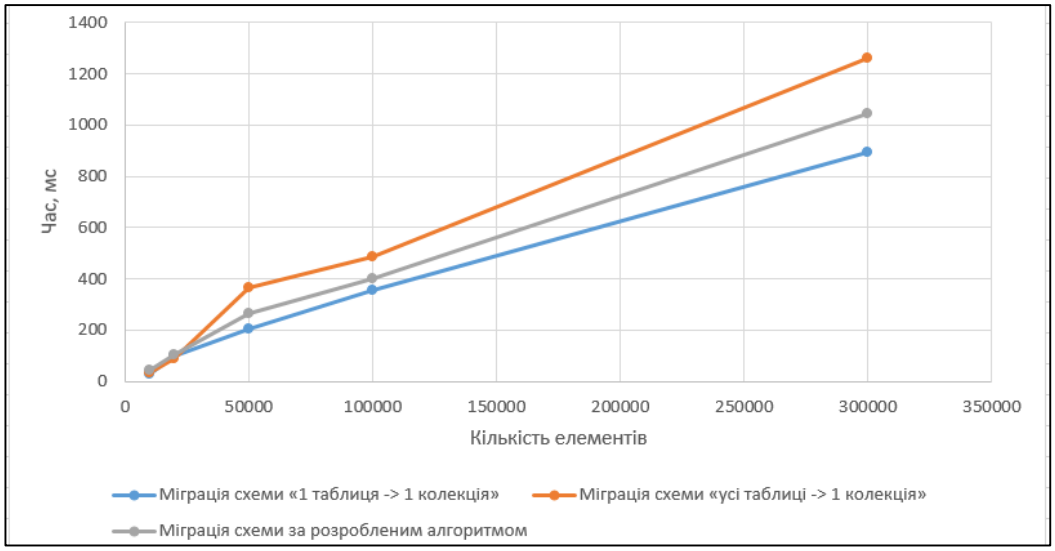

Рисунок 25 – Залежність часу виконання десятого запиту від методу міграції схеми та кількості елементів

Query11 = { ProductName, DishName, IngredientWeigth, IngredientDishId, DishDishId, IngredientProductId, ProductProductId }. В таблиці 12 відображено час виконання одинадцятого запиту для різної кількості даних для різних методів міграції схеми даних.

Таблиця 12 – Час виконання одинадцятого запиту

Кількість елементів у таблицях	Час виконання запиту, мс		
	Міграція схеми «1 таблиця -> 1 колекція»	Міграція схеми «усі таблиці -> 1 колекція»	Міграція схеми за розробленим алгоритмом
10000	56	34	59
20000	95	90	101
50000	213	364	197
100000	415	673	389
300000	1015	1478	969

Рисунок 26 відображає залежність часу виконання одинадцятого запиту від кількості елементів для різних алгоритмів.

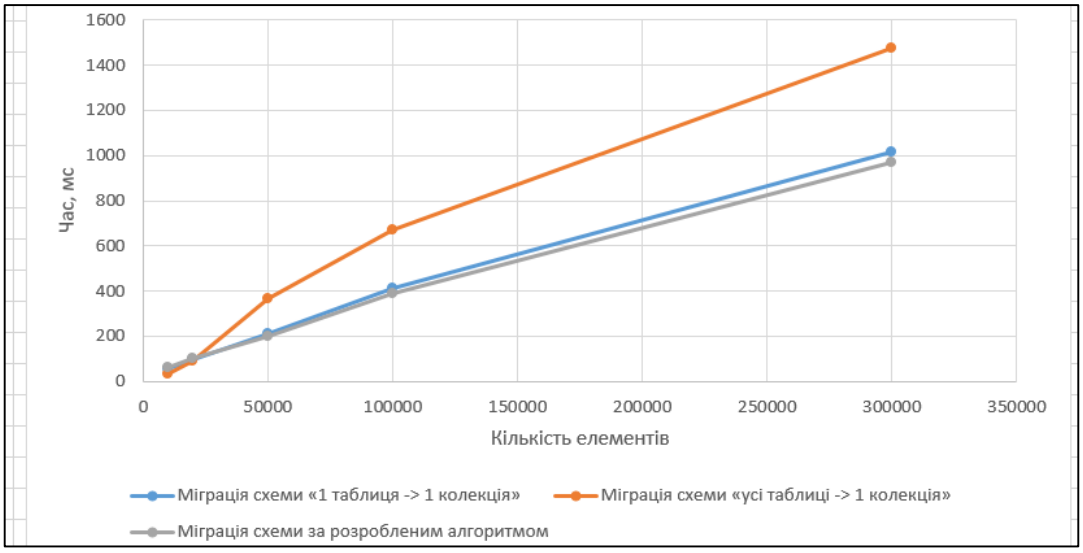

Рисунок 26 – Залежність часу виконання одинадцятого запиту від методу міграції схеми та кількості елементів

Query12 = { CartCustomerId, CartDeliveryAddressId, MenuItemCost, CartItemCount, CartItemMenuItemId, MenuItemMenuItemId, CartItemCartId }. В таблиці 13 відображено час виконання дванадцятого запиту для різної кількості даних для різних методів міграції схеми даних.

Таблиця 13 – Час виконання дванадцятого запиту

Кількість елементів у таблицях	Час виконання запиту, мс		
	Міграція схеми «1 таблиця -> 1 колекція»	Міграція схеми «усі таблиці -> 1 колекція»	Міграція схеми за розробленим алгоритмом
10000	115	103	76
20000	195	209	128
50000	430	694	365
100000	683	823	489
300000	1587	1745	1317

Рисунок 27 відображає залежність часу виконання дванадцятого запиту від кількості елементів для різних алгоритмів.

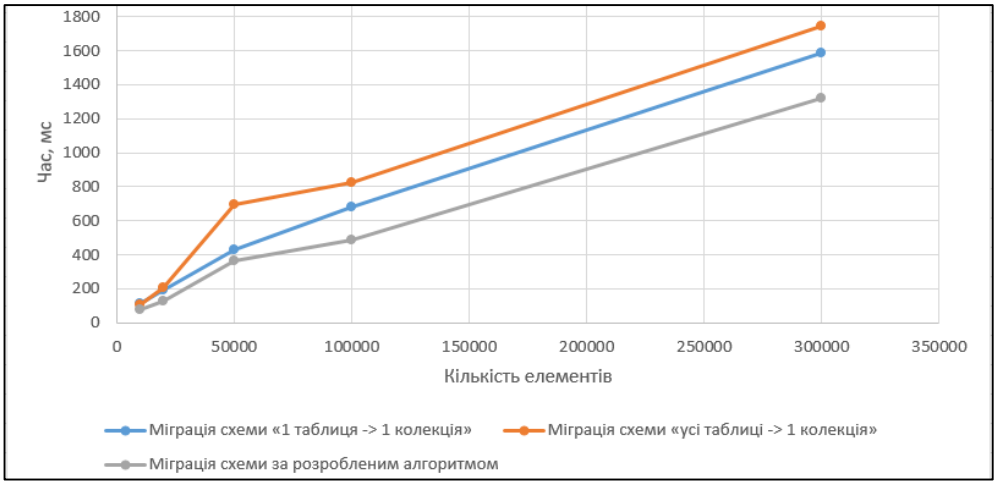

Рисунок 27 – Залежність часу виконання дванадцятого запиту від методу міграції схеми та кількості елементів

Query13 = { CartCustomerId, CartDeliveryAddressId, MenuItemCost, CartItemCount, CartItemMenuItemId, MenuItemMenuItemId, CartItemCartId, MenuItemDishId, CartDeliveryDateTime, DishName }. В таблиці 14 відображено час виконання тринадцятого запиту для різної кількості даних для різних методів міграції схеми даних.

Таблиця 14 – Час виконання тринадцятого запиту

Кількість елементів у таблицях	Час виконання запиту, мс		
	Міграція схеми «1 таблиця -> 1 колекція»	Міграція схеми «усі таблиці -> 1 колекція»	Міграція схеми за розробленим алгоритмом
10000	153	102	122
20000	285	201	243
50000	568	501	489
100000	895	872	772
300000	2392	3153	1953

Рисунок 28 відображає залежність часу виконання тринадцятого запиту від кількості елементів для різних алгоритмів.

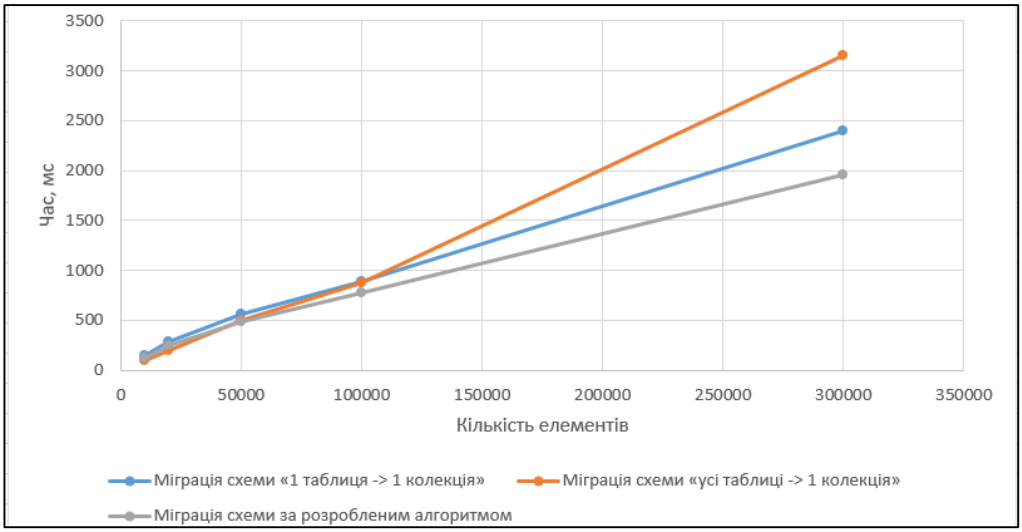

Рисунок 28 – Залежність часу виконання тринадцятого запиту від методу міграції схеми та кількості елементів

Query14 = { CartCustomerId, MenuItemCost, CartItemCount, CartItemMenuItemId, MenuItemMenuItemId, CartItemCartId, MenuItemDishId, CustomerCustomerId, CustomerBalance }. В таблиці 15 відображено час виконання чотирнадцятого запиту для різної кількості даних і різних методів міграції схеми.

Таблиця 15 – Час виконання чотирнадцятого запиту

Кількість елементів у таблицях	Час виконання запиту, мс		
	Міграція схеми «1 таблиця -> 1 колекція»	Міграція схеми «усі таблиці -> 1 колекція»	Міграція схеми за розробленим алгоритмом
10000	142	85	109
20000	215	139	184
50000	523	501	423
100000	1264	1158	784
300000	2589	3753	1835

Рисунок 29 відображає залежність часу виконання чотирнадцятого запиту від кількості елементів для різних алгоритмів.

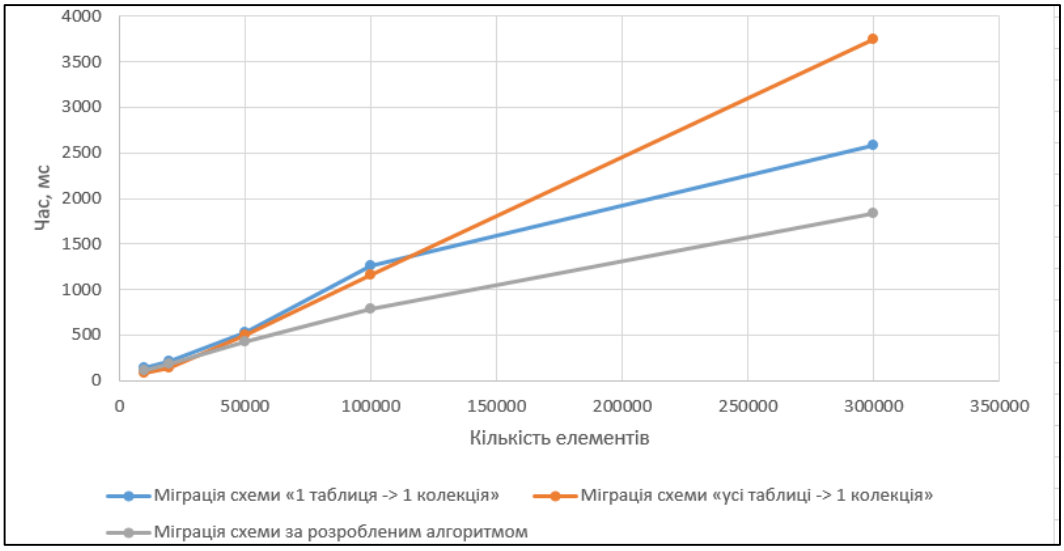

Рисунок 29 – Залежність часу виконання чотирнадцятого запиту від методу міграції схеми та кількості елементів

Query15 = { CartItemCount, CartItemMenuItemId, MenuItemMenuItemId, CartItemCartId, CartCustomerId, CartDeliveryDateTime }. В таблиці 16 відображено час виконання п’ятнадцятого запиту для різної кількості даних для різних методів міграції схеми даних.

Таблиця 16 – Час виконання п’ятнадцятого запиту

Кількість елементів у таблицях	Час виконання запиту, мс		
	Міграція схеми «1 таблиця -> 1 колекція»	Міграція схеми «усі таблиці -> 1 колекція»	Міграція схеми за розробленим алгоритмом
10000	112	45	46
20000	183	101	75
50000	577	409	236
100000	1083	1635	503
300000	2009	4163	905

Рисунок 30 відображає залежність часу виконання п’ятнадцятого запиту від кількості елементів для різних алгоритмів.

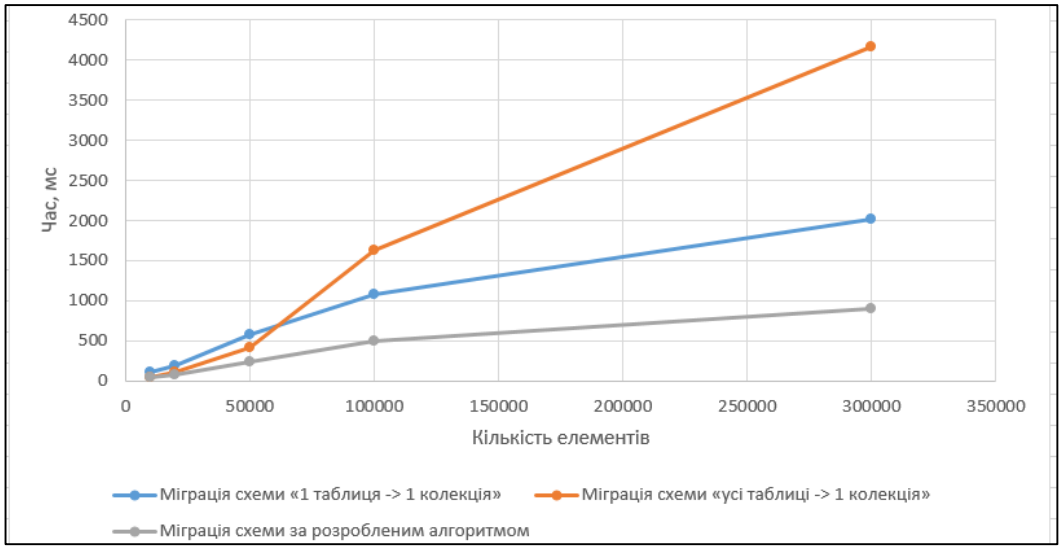

Рисунок 30 – Залежність часу виконання п’ятнадцятого запиту від методу міграції схеми та кількості елементів

Query16 = { DishName, DishDescription, CuisineName, CuisineDescription, CuisineCuisineId, DishCuisineId }. В таблиці 17 відображено час виконання шістнадцятого запиту для різної кількості даних для різних методів міграції схеми даних.

Таблиця 17 – Час виконання шістнадцятого запиту

Кількість елементів у таблицях	Час виконання запиту, мс		
	Міграція схеми «1 таблиця -> 1 колекція»	Міграція схеми «усі таблиці -> 1 колекція»	Міграція схеми за розробленим алгоритмом
10000	57	32	54
20000	73	55	76
50000	156	121	159
100000	335	487	342
300000	963	1546	967

Рисунок 31 відображає залежність часу виконання шістнадцятого запиту від кількості елементів для різних алгоритмів.

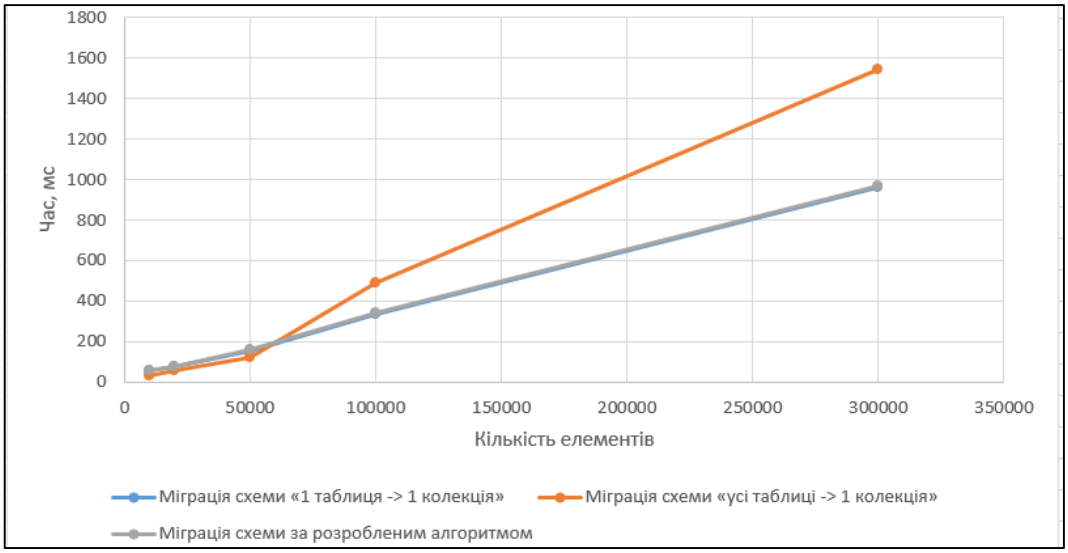

Рисунок 31 – Залежність часу виконання шістнадцятого запиту від методу міграції схеми та кількості елементів

Query17 = { RestaurantRestaurantId, RestaurantName, RestaurantPhoneNumber, RestaurantEmail, RestaurantScore, RestaurantURL, RestaurantCountry, RestaurantCity, RestaurantStreet, RestaurantHouseNumber }. В таблиці 18 відображено час виконання сімнадцятого запиту для різної кількості даних для різних методів міграції схеми.

Таблиця 18 – Час виконання сімнадцятого запиту

Кількість елементів у таблицях	Час виконання запиту, мс		
	Міграція схеми «1 таблиця -> 1 колекція»	Міграція схеми «усі таблиці -> 1 колекція»	Міграція схеми за розробленим алгоритмом
10000	63	61	87
20000	132	130	188
50000	534	589	679
100000	983	1635	1463
300000	1651	3106	2256

Рисунок 32 відображає залежність часу виконання сімнадцятого запиту від кількості елементів для різних алгоритмів.

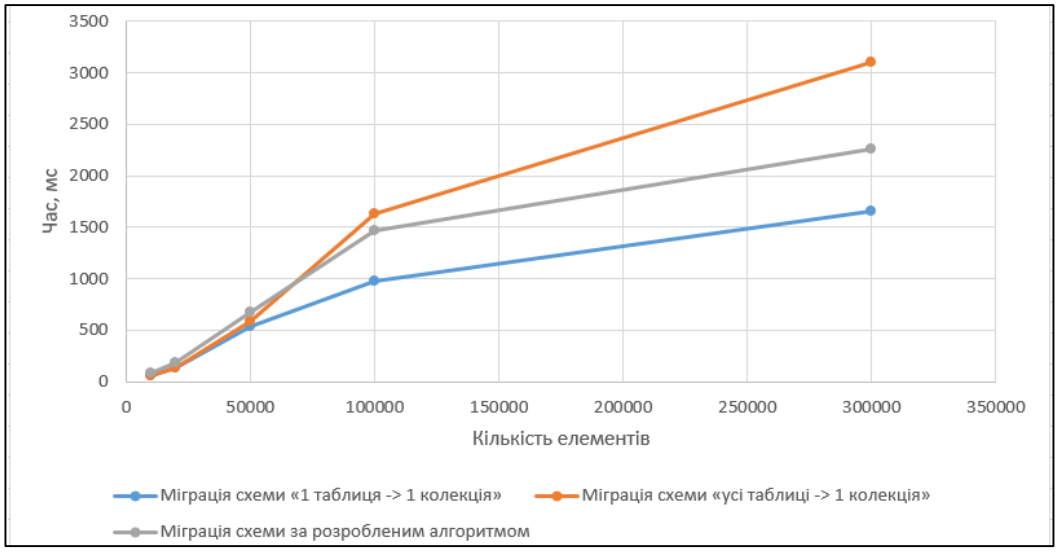

Рисунок 32 – Залежність часу виконання сімнадцятого запиту від методу міграції схеми та кількості елементів

Query18 = { DeliveryAddressDeliveryAddressId, DeliveryAddressCountry, DeliveryAddressCity, DeliveryAddressStreet, DeliveryAddressHouseNumber, DeliveryAddressFlatNumber }. В таблиці 19 відображено час виконання вісімнадцятого запиту для різної кількості даних для різних методів міграції схеми.

Таблиця 19 – Час виконання вісімнадцятого запиту

Кількість елементів у таблицях	Час виконання запиту, мс		
	Міграція схеми «1 таблиця -> 1 колекція»	Міграція схеми «усі таблиці -> 1 колекція»	Міграція схеми за розробленим алгоритмом
10000	70	71	73
20000	133	104	128
50000	269	218	272
100000	488	624	492
300000	1042	3415	1049

Рисунок 33 відображає залежність часу виконання вісімнадцятого запиту від кількості елементів для різних алгоритмів.

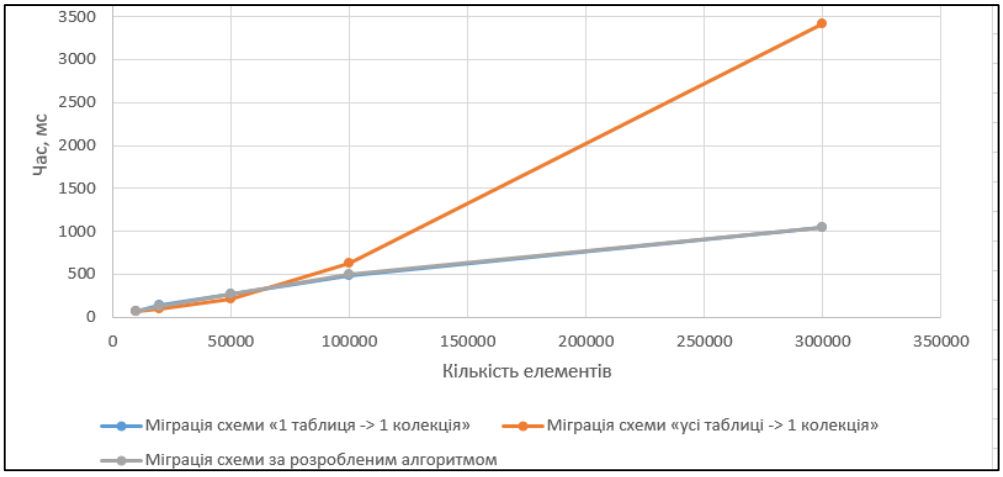

Рисунок 33 – Залежність часу виконання вісімнадцятого запиту від методу міграції схеми та кількості елементів

Query19 = { DeliveryAddressCountry, DeliveryAddressCity, DeliveryAddressStreet, DeliveryAddressHouseNumber, DeliveryAddressFlatNumber, CartCustomerId, CartDeliveryAddressId, DeliveryAddressDeliveryAddressId }. В таблиці 20 відображено час виконання дев’ятнадцятого запиту для різної кількості даних для різних методів міграції схеми даних.

Таблиця 20 – Час виконання дев’ятнадцятого запиту

Кількість елементів у таблицях	Час виконання запиту, мс		
	Міграція схеми «1 таблиця -> 1 колекція»	Міграція схеми «усі таблиці -> 1 колекція»	Міграція схеми за розробленим алгоритмом
10000	63	40	59
20000	80	68	86
50000	223	188	204
100000	418	378	408
300000	1236	3012	1193

Рисунок 35 відображає залежність часу виконання дев’ятнадцятого запиту від кількості елементів для різних алгоритмів.

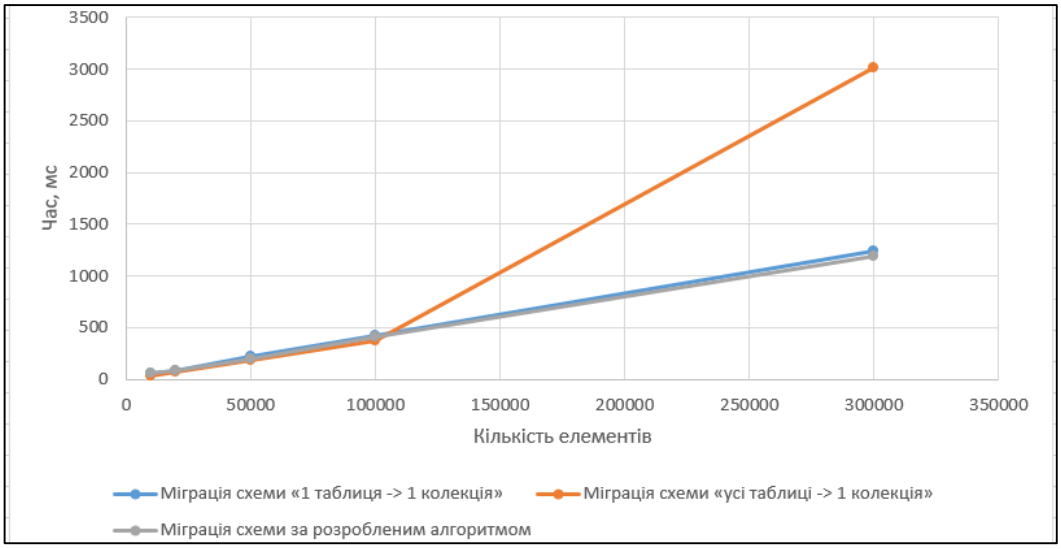

Рисунок 35 – Залежність часу виконання дев’ятнадцятого запиту від методу міграції схеми та кількості елементів

Query20 = { RestaurantRestaurantId, RestaurantName, RestaurantPhoneNumber, RestaurantEmail, RestaurantScore, RestaurantURL }. В таблиці 21 відображено час виконання двадцятого запиту для різної кількості даних для різних методів міграції схеми даних.

Таблиця 21 – Час виконання двадцятого запиту

Кількість елементів у таблицях	Час виконання запиту, мс		
	Міграція схеми «1 таблиця -> 1 колекція»	Міграція схеми «усі таблиці -> 1 колекція»	Міграція схеми за розробленим алгоритмом
10000	30	34	32
20000	60	55	58
50000	136	120	135
100000	247	225	243
300000	636	519	632

Рисунок 36 відображає залежність часу виконання двадцятого запиту від кількості елементів для різних алгоритмів.

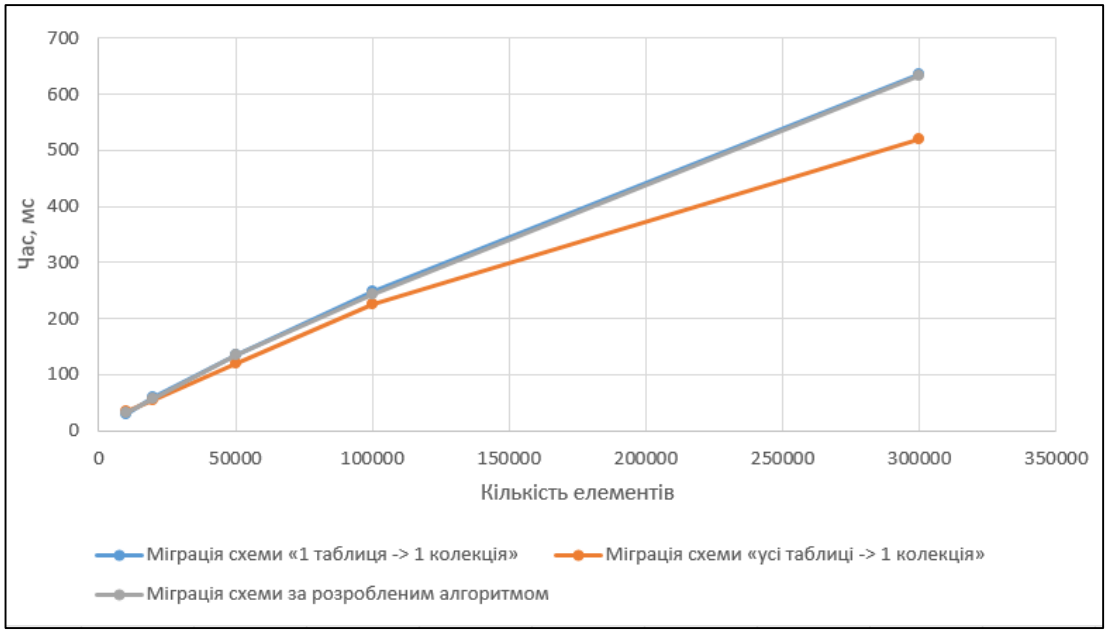

Рисунок 36 – Залежність часу виконання двадцятого запиту від методу міграції схеми та кількості елементів

Таким чином, було проведено дослідження для усіх запитів на різних об’ємах баз даних.

5.5 Аналіз результатів дослідження

Аналізуючи графіки залежності часу виконання запитів для різних варіантів схем даних при різній кількості збережених даних, можна сказати, що в переважній кількості запитів різниця у часі виконання стає більш істотною та, відповідно, об’єктивніше відображує ефективність використання того чи іншого методу міграції схеми даних, при збільшені кількості даних у таблицях бази даних. Ще одним підтвердженням об’єктивності алгоритму на більшій кількості елементів є те, що зазвичай міграції даних з реляційних на документо-орієнтовні моделі є пріоритетним завданням для великих комерційних систем, де зберігаються великі об’єми даних.

В таблиці 22 наведено середній час виконання запитів для трьох схем бази даних при різній кількості елементів у кожній таблиці.

Таблиця 22 – Середній час виконання запитів для трьох схем бази даних при різній кількості елементів у кожній таблиці

Кількість елементів у таблицях	Середній час виконання запитів, мс		
	Міграція схеми «1 таблиця -> 1 колекція»	Міграція схеми «усі таблиці -> 1 колекція»	Міграція схеми за розробленим алгоритмом
10000	86,82	66	75,57
20000	163,35	131,19	140,95
50000	353,48	383,71	301,81
100000	693,05	871,67	583,9
300000	1812,52	2606,9	1488,76

На рисунку 37 зображено гістограму з усередненим часом виконання запитів для трьох схем бази даних при різній кількості елементів у кожній таблиці.

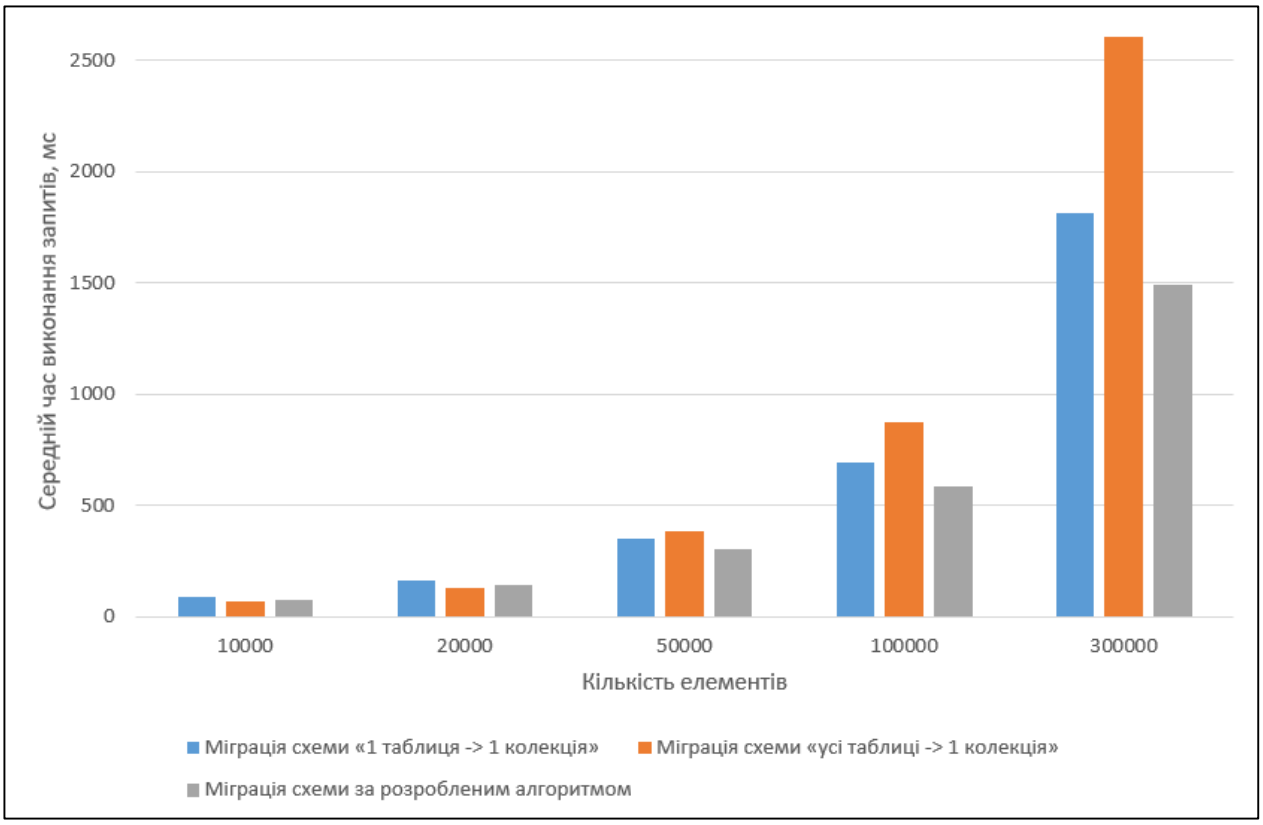

Рисунок 37 – Гістограма усередненого часу виконання запитів для трьох схем бази даних при різній кількості елементів у кожній таблиці

При порівняно невеликій кількості даних на усіх трьох схемах даних запити відпрацьовують за малий проміжок часу, причому схема, побудована за методом міграції «усі таблиці -> 1 колекція» дає кращі результати, це можна пояснити тим, що усі дані знаходяться в одній колекції, тобто не треба додатково зв'язувати окремі колекції задля отримання результату; схема, побудована за допомогою міграції «1 таблиця -> 1 колекція» дає дещо гірші результати, ніж схема, побудована за допомогою алгоритму міграції на основі запитів, розробленому в рамках даної роботи, бо у більшості випадків схема після міграції «1 таблиця -> 1 колекція» передбачає зв'язування більшої кількості таблиць для отримання результатів певного запиту, аніж у випадку схеми, отриманої за алгоритмом на основі запитів.

Для 10000 елементів виявилося, що у базі даних зі схемою «усі таблиці -> 1 колекція» запити виконуються на 23,97 % швидше, аніж у базі даних зі схемою «1 таблиця -> 1 колекція», в той же час у базі даних зі схемою, побудованою за

допомогою алгоритму на основі запитів, запити виконуються на 12,95 % швидше, ніж у базі даних зі схемою «1 таблиця -> 1 колекція».

Для бази даних з 20000 елементами виявилося, що на схемі «усі таблиці -> 1 колекція» запити виконуються на 15,67 % швидше, аніж на схемі «1 таблиця -> 1 колекція», в той же час на схемі, побудованій за допомогою алгоритму на основі запитів, запити виконуються на 9,4 % швидше, ніж на схемі «1 таблиця -> 1 колекція».

Зі збільшенням кількості даних в базі даних прослідковується тенденція до погіршення показників для схеми «усі таблиці -> 1 колекція», це пов'язано з тим, що така колекція містить великий об'єм даних, внаслідок чого проявляються проблеми з пам'яттю в СКБД і запити відпрацьовують повільніше.

В той самий час, дослідження для схеми, розробленої за алгоритмом на основі запитів, на великій кількості даних дає у переважній кількості запитів стабільні результати, час виконання запитів є найменшим у порівнянні з іншими методами.

Для 50000 елементів виявилося, що у базі даних зі схемою «1 таблиця -> 1 колекція» запити виконуються на 7,87 % швидше, аніж у базі даних зі схемою «усі таблиці -> 1 колекція», в той же час у базі даних зі схемою, побудованою за допомогою алгоритму на основі запитів, запити виконуються на 21,34 % швидше, ніж у базі даних зі схемою «усі таблиці -> 1 колекція», у базі даних зі схемою, побудованою за допомогою алгоритму на основі запитів, запити виконуються на 14,7 % швидше, аніж у базі даних зі схемою «1 таблиця -> 1 колекція».

Для 100000 елементів виявилося, що у базі даних зі схемою «1 таблиця -> 1 колекція» запити виконуються на 20,49 % швидше, аніж у базі даних зі схемою «усі таблиці -> 1 колекція», в той же час у базі даних зі схемою, побудованою за допомогою алгоритму на основі запитів, запити виконуються на 33,01 % швидше, ніж у базі даних зі схемою «усі таблиці -> 1 колекція», у базі даних зі схемою, побудованою за допомогою алгоритму на основі запитів, запити виконуються на 15,75 % швидше, аніж у базі даних зі схемою «1 таблиця -> 1 колекція».

Для 300000 елементів виявилося, що у базі даних зі схемою «1 таблиця -> 1

колекція» запити виконуються на 30,47 % швидше, аніж у базі даних зі схемою «усі таблиці -> 1 колекція», в той же час у базі даних зі схемою, побудованою за допомогою алгоритму на основі запитів, запити виконуються на 42,89 % швидше, ніж у базі даних зі схемою «усі таблиці -> 1 колекція» , у базі даних зі схемою, побудованою за допомогою алгоритму на основі запитів, запити виконуються на 17,86 % швидше, аніж у базі даних зі схемою «1 таблиця -> 1 колекція».

Після аналізу результатів для різних об’ємів даних можна зробити висновок, що для невеликої за обсягом бази даних (невелика кількість таблиць та даних) цілком доцільно використовувати метод «усі таблиці -> 1 колекція» для міграції схеми даних з реляційної моделі до документо-орієнтовної, адже він дозволяє запобігти виконанню операцій зв’язування колекцій, які є ресурсозатратними в MongoDB. Для бази даних з невеликою кількістю таблиць доцільним є використання методу «1 таблиця -> 1 колекція» для міграції схеми, адже цей метод є найпростішим та не вимагає додаткової роботи над перепроектуванням схеми, в той самий час, якщо схема містить велику кількість таблиць, то такий метод показує гірші результати, так як задля виконання запитів відпрацьовує значна кількість операцій зв’язування колекцій. Якщо мова йде про схему даних з великою кількістю таблиць та з високим рівнем наповненості, то найкращим варіантом є перепроектування схеми даних за допомогою розробленого в рамках даної роботи алгоритму міграції схеми даних на основі запитів, адже в результаті такого перепроектування кількість таблиць, необхідних для виконання запита, зменшена, тобто відбувається істотно менше операцій зв’язування таблиць, та разом з тим, таблиці містять менше надлишкової інформації для запита, тобто переважно із певної таблиці в запиті задіяна більша частина полів.

ВИСНОВКИ

В ході виконання кваліфікаційної роботи було:

– розглянуто теоретичні аспекти міграції баз даних, пов’язані терміни та процеси;

– проаналізовано популярність та актуальність використання нереляційних баз даних, на основі цього аналізу обрано нереляційну документо-орієнтовну базу даних MongoDB з json-подібною моделлю зберігання даних для розгляду у якості цільової бази даних при міграції з реляційної моделі;

– виконано аналітичний огляд методів і стратегій міграції між реляційними моделями та json-подібними моделями MongoDB, наведено особливості таких методів, обмеження таких методів, випадки доцільності використання та недоліки таких методів;

– розглянуто можливість використання реляційної алгебри і теорії множин у контексті моделей даних та запитів, а також при перепроектуванні моделей;

– обрано стратегію міграції моделей даних, яка передбачає перепроектування схеми бази даних у відповідності до запитів до бази даних;

– створено математичну модель та описано відповідний алгоритм міграції даних між реляційною та документо-орієнтовною базами з використанням теорії множин за принципами вищезазначеної стратегії;

– розроблено програмне забезпечення для підтримки міграції спроектованим алгоритмом (за допомогою .NET Core);

– спроектовано тестову реляційну схему даних (11 сутностей) з предметної області закладів харчування (для сайту-агрегатору ресторанів з можливістю зробити онлайн-замовлення) та запити до неї (20 запитів), створено базу даних MS SQL за цією схемою;

– виконано міграцію схеми даних за допомогою розробленого програмного

забезпечення;

– виконано міграцію даних для результуючої схеми даних та для двох інших типів міграції схем («1 таблиця -> 1 колекція» та «усі таблиці -> 1 колекція»), виміряно час виконання міграції для різної кількості даних;

– за допомогою менеджеру підключень MongoDB – Robo 3T – виміряно час виконання кожного з двадцяти розроблених запитів (по 10 спроб задля зменшення похибки вимірювань) до усіх трьох варіантів цільової бази даних при різній кількості збережених елементів (10000, 20000, 50000, 100000 та 300000 елементів).

Результатом дослідження в рамках кваліфікаційної роботи є алгоритм гетерогенної модельно-неоднорідної міграції даних з реляційної до документо-орієнтовної схеми даних, розроблений з урахуванням запитів до бази даних, а також програмне забезпечення, що автоматизує розроблений алгоритм та здійснює міграцію даних з MS SQL до MongoDB.

В рамках дослідження були отримані результати щодо швидкості міграції даних до різних цільових схем даних: варіант схеми «усі таблиці -> 1 колекція» виявився найшвидшим, варіант «1 таблиця -> 1 колекція» показав середній показник, а міграція на схему, отриману в результаті розробленого в рамках роботи алгоритма показала найдовший час виконання, це пов’язане з кількістю операцій додавання даних на цільові схеми та складністю запитів вибірки даних у вихідній базі даних.

Переважання часу міграції даних не є показником якості спроектованої схеми даних, так як міграція є одноразовим процесом, більш вагомим показником є швидкість виконання запитів до цільової бази даних, адже під час використання цільової бази даних вони виконуватимуться постійно. Після порівняння середніх результатів часу виконання запитів на різних варіантах схем даних при різній кількості даних в базах виявилося, при відносно невеликих об’ємах бази даних міграція «усі таблиці -> 1 колекція» дає найкращі результати, міграція «1 таблиця -> 1 колекція» дає дещо гірші результати, схема, отримана в результаті розробленого алгоритму відпрацьовує найдовше, але зі збільшенням кількості

даних в базі показники змінюються: запити на схемі, отриманій в результаті розробленого алгоритму відпрацьовують найшвидше у порівнянні з іншими схемами. Усереднені дані за типом схеми даних (середній час при усіх об’ємах бази даних) показують, що найшвидше запити виконуються на схемі, отриманій в результаті розробленого алгоритму на основі запитів, середні показники показала схема «1 таблиця -> 1 колекція», найдовшим час виявився при роботі зі схемою «усі таблиці -> 1 колекція».

Отже, можна зробити висновок, що отриманий алгоритм міграції схеми даних та розроблене програмне забезпечення для міграції даних є придатними для використання на реальних прикладах, а також є об’єктами для подальших досліджень і можливих удосконалень.

Робота пройшла апробацію на декількох наукових конференціях, а саме: на XV Всеукраїнській науково-практичній WEB конференції аспірантів, студентів та молодих вчених «Комп’ютерні інтелектуальні системи та мережі» (додаток Е) та на Дванадцятій міжнародній науково-технічній конференції «Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління» (додаток Ж). Також матеріали статті по даній роботі прийняті до друку в рецензований науковий журнал відкритого доступу з переліку наукових фахових видань України (за спеціальністю 121 Інженерія програмного забезпечення) «Сучасний стан наукових досліджень та технологій в промисловості» (додаток И).

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. International Roadmap for Devices and Systems. More Moore White Paper: веб-сайт. URL: https://irds.ieee.org/images/files/pdf/2016_MM.pdf (дата звернення: 25.03.2022).

2. Morris J. Practical Data Migration. BCS, The Chartered Institute for IT, London, 2012, 266 с.

3. Homogeneous vs Heterogeneous migration: веб-сайт. URL: https://rtfm.co.ua/aws-database-migration-service-obzor-i-primer-migracii-self-hosted-mariadb-v-aws-aurora-rds/#Homogeneous_vs_Heterogeneous_migration (дата звернення: 30.03.2022).

4. Preston Z. Practical Guide to Large Database Migration. CRC Press, USA, 2021, 198 с.

5. Andreas M. Providing Database Migration Tools. A Practitioner's Approach. *21st International Conference on Very Large Data Bases (VLDB)*. 2015. С. 635 – 641.

6. Ji L. F., Azmi N. F. M. The development of a new data migration model for NOSQL databases with different schemas in environment management system. *Journal of Environmental Treatment Techniques*. 2020. № 8 (2). С. 787 – 793.

7. Kuzochkina A., Shirokopetleva M., Dudar Z. Analyzing and Comparison of NoSQL DBMS. *2018 International Scientific-Practical Conference on Problems of Infocommunications Science and Technology*. 2018. С. 560 – 564. DOI: https://doi.org/10.1109/INFOCOMMST.2018.8632133

8. DB-Engines Ranking: веб-сайт. URL: https://db-engines.com/en/ranking (дата звернення: 10.04.2022).

9. Chickerur S.; Goudar A.; Kinnerkar A. Comparison of Relational Database with Document-Oriented Database (MongoDB) for Big Data Applications. *2015 8th International Conference on Advanced Software Engineering & Its Applications (ASEA)*. 2015. С. 41 – 47. DOI: https://doi.org/10.1109/ASEA.2015.19

10. Mazurova O., Naboka A., Shirokopetleva M. Research of ACID transaction

implementation methods for distributed databases using replication technology. *Innovative technologies and scientific solutions for industries*. 2021. № 2 (16). С. 1 – 5. DOI: 10.30837/ITSSI.2021.16.019

11. Степовик А. Н, Ефанова Н. В. Анализ реляционных и нереляционных баз данных. Цифровизация экономики: направления, методы инструменты. 2019. С. 414 – 416.

12. Ceresnak R., Dudas A., Matiasko K. Mapping rules for schema transformation: SQL to NoSQL and back. *2021 International Conference on Information and Digital Technologies*. 2021. С. 52 – 58. DOI: https://doi.org/10.1109/IDT52577.2021.9497629

13. Hanine M., Bendarag A., Boutkhoum O. Data Migration Methodology from Relational to NoSQL Databases. *International Journal of Computer, Electrical, Automation, Control and Information, Engineering*. 2015. № 9 (12). С. 2566 – 2570.

14. Alalfi M. H.. Automated Algorithm for Data Migration from Relational to NoSQL Databases. *Al-Nahrain Journal for Engineering Sciences (NJES)*. 2018. № 21 (1). С. 60 – 65. DOI: https://doi.org/10.29194/NJES2101

15. Fouad T., Mohamed B. Model transformation from object relational database to NoSQL document database. *NISS19*. 2019. № 49. С. 1 – 5. DOI: https://doi.org/10.1145/3320326.3320381

16. Li X., Ma Z., Chen H. QODM: A Query-Oriented Data Modeling Approach for NoSQL Databases. *IEEE Workshop on Advanced Research and Technology in Industry Applications*. 2014. С. 338 – 345.

17. Alotaibi O., Pardede E. Transformation of Schema from Relational Database (RDB) to NoSQL Databases. *Data*. 2019. № 4 (4). С. 148. DOI: https://doi.org/10.3390/data4040148

18. Ain El Hayat S., Bahaj M. Modeling and transformation from temporal object relational database into mongodb: Rules. *Advances in Science, Technology and Engineering Systems*. 2020. № 5 (4). С. 618–625. DOI: https://doi.org/10.25046/aj050473

19. Mason R. T. NoSQL databases and data modeling techniques for a document-oriented NoSQL database. *Informing Science & IT Education Conference (InSITE)*. 2015

С. 259 – 268. DOI: https://doi.org/10.28945/2245

20. Alekseev A. A., Osipova V. V., Ivanov M. A. Efficient data management tools for the heterogeneous big data warehouse. Physics of Particles and Nuclei Letters. 2016. № 13 (5). С. 689 – 692. DOI: https://doi.org/10.1134/S1547477116050022

21. Gu Y.; Wang X.; Shen S. Analysis of data storage mechanism in NoSQL database MongoDB. *2015 IEEE International Conference on Consumer Electronics*. 2015. С. 158 – 159.

22. Dabowsa N. I., Maatuk A. M., Elakeili S. M. Converting Relational Database to Document-Oriented NoSQL Cloud Database. *2021 IEEE 1st International Maghreb Meeting of the Conference on Sciences and Techniques of Automatic Control and Computer Engineering MI-STA*. 2021. С. 381 – 386. DOI: https://doi.org/10.1109/MI-STA52233.2021.9464488

23. Date C. J. SQL and Relational Theory: How to Write Accurate SQL Code. O'Reilly Media, 2012. 448 с.

24. Перетятко М. В., Широкопетлєва М. С. Стратегії міграції між реляційними і документними моделями зберігання даних. *XV Всеукраїнська науково-практична WEB конференція аспірантів, студентів та молодих вчених «Комп'ютерні інтелектуальні системи та мережі»*: матеріали Всеукр. наук.-практ. конф., м. Кривий Ріг, 22 – 24 берез. 2022 р. / Криворізький нац. ун. Кривий Ріг, 2022. С. 76 – 78.

25. Перетятко М. В., Широкопетлєва М. С. Стратегії міграції між реляційними і документними моделями зберігання даних. *Дванадцята міжнародна науково-технічна конференція «Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління»*: тези доп. міжнар. наук-практ. конф, 27 – 28 квітня 2022 р. Баку – Харків – Жиліна, 2022. С. 136.

26. Peretiatko M., Shirokopetleva M., Lesna N. Research of methods to support data migration between relational and document data storage models. *Innovative Technologies and Scientific Solutions for Industries*. 2022. № 2 (20). (прийнято до друку).