МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження методів порціювання даних для масштабування програмних систем

(тема)

Виконав:
Студент 2 курсу, групи ІПЗм-21-1
Ковалевич Богдан Ігорович
(прізвище, ініціали)

Спеціальність 121 - Інженерія програмного забезпечення
(код і повна назва спеціальності)

Тип програми Освітньо-наукова
(освітньо-професійна або освітньо-наукова)

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

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

2023 р.

Харківський національний університет радіоелектроніки
Факультет Комп'ютерних наук
(повна назва)

Кафедра Програмної інженерії
(повна назва)

Рівень вищої освіти другий (магістерський)
Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Інженерія програмного забезпечення
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____

(підпис)

«___»_____ 2023 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Ковалевичу Богдану Ігоровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження методів порціювання даних для масштабування програмних систем

Затверджена наказом університету від 29.03.2023 № 302ст

2. Вихідні данні до роботи електронні ресурси за вибраною тематикою, вимоги до реалізації методів порціювання даних для SQL та NoSQL СУБД, бази даних MS SQL Server, Cosmos DB, MongoDB, MySQL, PostgreSQL, середовища Azure Cosmos DB Explorer, SQL Management Studio, Visual Studio 2022, мови T-SQL, C#.

3. Перелік питань, що потрібно опрацювати в роботі вступ, аналіз предметної області, розробка структур баз даних для досліду, проектування різного роду запитів для досліду, розробка середовищ для отримання замірів,проведення експерименту, порівняння обраних методів, формування рекомендацій.

4. Перелік графічного матеріалу із зазначенням креслеників, схем, слайдів, ілюстрацій актуальність області дослідження, мета, постановка задачі, аналіз проблемної області, етапи проектування планування експерименту, проектування БД, архітектура і реалізація програмного забезпечення, результати, рекомендації, висновки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Позначка про виконання
1	Аналіз проблемної області дослідження	23.01.23 – 07.02.23	Виконано
2	Розробка постановки задачі	08.02.23 – 14.02.23	Виконано
3	Дослідження існуючих методів порціювання даних	15.02.23 – 18.02.23	Виконано
4	Планування експериментального дослідження	19.02.23 – 24.02.23	Виконано
5	Проектування схеми БД	25.02.23 – 01.03.23	Виконано
6	Проектування середовища для експерименту	01.03.23 – 14.03.23	Виконано
7	Розробка запитів	15.03.23 – 01.04.23	Виконано
8	Проведення експерименту та аналіз результатів	02.04.23 – 20.04.23	Виконано
9	Оформлення статті або тез доповіді	17.04.23 – 23.04.23	Виконано
10	Підготовка пояснювальної записки	01.04.23 – 26.04.23	Виконано
11	Підготовка презентації та доповіді	26.04.23 – 27.04.23	Виконано
12	Нормоконтроль	28.04.23 – 02.05.23	Виконано
13	Рецензування	03.05.23 – 06.05.23	Виконано
14	Занесення диплома в електронний архів	07.05.23	Виконано
15	Попередній захист	07.05.23	Виконано
16	Допуск до захисту у зав. кафедри	08.05.23	Виконано

Дата видачі завдання 23 січня 2023р.

Студент _____ Ковалевич Б. І

(підпис) (прізвище, ініціали)

Керівник роботи _____ доц.кафедри ПІ Мазурова О.О.

(підпис) (прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Кваліфікаційна робота магістра містить: 94 с, 28 рис., 22 табл., 21 джер.

БАЗА ДАНИХ, ЗАМІР ЕФЕКТИВНОСТІ, МЕТРИКА ЯКОСТІ, МОВА ПРОГРАМУВАННЯ C#, ОПТИМІЗАЦІЯ ЗАПИТУ, ПОРЦІЮВАННЯ, СУБД, COSMOSDB, MONGODB, MS SQL SERVER, MYSQL, NO-SQL, POSTGRESQL, SQL.

Об’єктом дослідження є методи порціювання даних для масштабування баз даних. Предмет дослідження пов’язаний з експериментальним дослідженням різного виду операцій над даними в SQL-орієнтованих та No-SQL системах управління базами даних та порівнянні ефективності відповідних методів порціювання.

Методами розробки та проектування є платформа .NET CORE 6, мови програмування C#, мови запитів T-SQL, за основу взяті СУБД MS SQL Server 2017 та CosmosDB V2 як основних представників SQL та NoSQL-орієнтованих систем, середовище розробки Visual Studio 2022, менеджер баз даних SQL Management Studio.

У результаті кваліфікаційної роботи була розроблена база даних та запити для дослідження, досліджено методи порціювання даних та складені рекомендації щодо ефективності виконання операцій над даними для конкретного методу.

C#, COSMOSDB, DATA BASES, DBMS, EFFICIENCY MEASUREMENT, MONGODB, MS SQL SERVER, MYSQL, NO-SQL, PARTITIONING, POSTGRESQL, QUALITY METRIC, REQUEST OPTIMISATION, SQL.

The purpose of the work is investigating the methods of data partitioning for scaling databases. The subject of the research is related to the experimental study of various types of operations on data in SQL-oriented and No-SQL database management systems and comparison of the effectiveness of the corresponding methods of partitioning.

Design methods and development are based on the .NET Core 6 Plaform, C# programing language, T-SQL query language, MS SQL Server 2017 DBMS, CosmosDB V2 as common SQL and NoSQL-oriented systems, Visual Studio 2022 IDE, SQL Management Studio as a database manager.

As a result of the work, a database and queries for research were developed, methods of data partitioning were investigated, and recommendations were made on the efficiency of operations on data for a specific method.

Умови публікації пояснювальної записки

Я, Ковалевич Богдан Ігорович, студент гр. ІПЗм-21-1, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Дослідження методів порціювання даних для масштабування програмних систем», 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 ознайомлений з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ВСТУП

Сучасні технології неможливі без використання комп’ютерів. Комп’ютерні технології запроваджуються через програмне забезпечення. Будь-яке програмне забезпечення використовує різноманітні рішення щодо зберігання даних. Кількість даних зростає щосекунди в світі. Наприклад, чорна п’ятниця може за день зібрати мільйони транзакцій, це стосується банкової системи, а також інтернет замовлень. Сама лише аналітика ютуб контенту займає терабайти, не говорячи вже про файли відео записів. Наразі кожне серйозне програмне забезпечення зіштовхується з проблемою масштабування даних. В силу обмеження апаратних засобів пам’яті машини масштабування допомагає розподілити дані на декілька носіїв. Порціювання даних є одним з засобів масштабування даних.

Застосунок чи програма невід’ємно пов’язані з використанням сховища даних, для зберігання та використання структурованих даних найбільш вживаною є реляційна технологія та реляційні бази даних. Також з популярністю використання хмарних застосувань починають займати нішу No-SQL бази даних (БД), перевагою яких є масштабування та як наслідок висока пропускна здатність для виконання операцій.

При роботі з великим об’ємом даних буде задіяно багато сегментів, та не всі фізично можуть розміщуватись в оперативному просторі машини, тому необхідно більше часу для синхронізації сегментів та процесу вигрузки оперативної пам’яті. Порціювання даних ставить за мету зменшити кількість сегментів даних при проведенні операції. Порціювання [1] може відбуватись між декількома машинами, такий метод є шардінгом. Програми мають дані, які логічно розділяються для різних запитів або операцій над даними, тому використання методів порціювання або шардингу є доцільним для оптимізації швидкості виконання операцій.

Оптимізації операцій даних шляхом методів порціювання не потребують збільшення кількості машин для опрацювання, порціювання відбувається за рахунок логічної розбивки даних на сектори та зменшення загального часу відклику операцій за рахунок зменшення даних для сканування.

Метою даної роботи є дослідження існуючих методів порціювання даних NoSQL та SQL баз даних, як методів масштабування баз даних. Порівняння їх продуктивності виконання операцій в області з обрахунками фінансових операцій та формуванням звітності в сфері процесів працевлаштування на базі бюджету держзамовлень.

Під час виконання кваліфікаційної роботи був проведений аналіз проблемної області реалізації методів порціювання даних на основі публікацій світових та вітчизняних фахівців в проблемній області (див. додаток А). Розглянуті основні методи порціювання даних на різних СУБД, а також обрані найбільш актуальні СУБД для проведення дослідження.

Також, було спроектовано процес проведення експерименту дослідження, що в свою чергу включає проектування схем та структур БД, запитів до цих баз. Описані обмеження проведення експерименту, сформовані метрики для порівняння ефективності методів порціювання на різних СУБД.

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

На основі плану проведення дослідження та його результатів було розроблено презентацію (див. додаток В). За результатами роботи були створені тези доповіді на 27-й Міжнародний молодіжний форум «Радіоелектроніка і молодь у XXI столітті» (див. додаток Г), а також, підготовлено для подачі наукову статтю “Дослідження методів порціювання даних для масштабування програмних систем”.

Також, кваліфікаційна робота перевірена на відповідність вимогам оформлення (див. додаток Д).

1 АНАЛІЗ ПРОБЛЕМНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз проблемної області дослідження

Оптимізація ефективності виконання операцій над даними є вагомим вкладом в бізнес, де важлива кожна мілісекунда відклику для клієнта. Для серйозних корпоративних проектів важливо враховувати кожну деталь при проектуванні та розробці застосунку. Одне з чутливих аспектів масштабування застосунку при можливому великому навантажені. Окрім великого навантаження важливим чинником є оптимізація процесу обробки запитів.

Це все дуже пов’язано з вибором сховища для застосунку, адже є різноманітні СУБД, які можуть по різному виконувати ті чи інші задачі, які можуть краще або гірше виконати вимоги застосунку. Часто буває так, що об’єм даних стає дуже великим, для обробки деяких запитів потрібно все більше часу.

Важливим процесом є масштабування. Завдяки цьому є приріст ефективності та здатності до обробки великого об’єму даних. В залежності від потреб бізнесу та технічних вимог, існує 2 основних типи масштабування баз даних: вертикальне та горизонтальне.

Горизонтальне масштабування БД проходить шляхом додавання додаткових серверів, кожен з яких оброблює свою частину даних. Цей метод дозволяє лінійно збільшувати потужність та продуктивність бази даних при збільшенні об’єму даних. Із плюсів можна виділити те, що система обробки може працювати паралельно, що дозволяє прискорити процес обробки та отримання результатів.

Верикальне масштабування - це процес збільшення продуктивності та потужності сервера бази даних шляхом додавання ресурсів, таких як процесори, оперативна пам’ять або сховище даних. Перевагою є простота в реалізації, потребує мінімальних змін в програмному забезпеченні, спрощення адміністрування. Із недоліків виділяють обмеженість масштабування, адже ресурси сервера можуть бути вичерпані, велика ціна, вертикальне масштабування може бути дорогим, особливо якщо необхідно потужне обладнання.

Методами горизонтального масштабування є шардінг, реплікування та порціювання. Шардінг - це процес розділення великої бази даних на менші фрагменти, які називаються шардами. Кожен шард містить певний діапазон даних та зберігається на окремому сервері. Реплікування - це процес створення даних на кількох серверах [2]. Якщо один сервер вийде з ладу, інший сервер може продовжувати обробку запитів, що забезпечує більшу надійність та доступність. Крім того реплікування може бути використано для розподілу навантаження системи. Порціювання - це процес розділення в базі даних на менші фрагменти, які зберігаються на різних серверах. Ключовою різницею з шардінгом є способ розділення даних та зберігання їх на сервері. Шарди зазвичай не перетинаються, а от фрагменти даних при порціювання можуть перетинатися, тобто один елемент даних може бути розподілений між декількома фрагментами.

Мета методів порціювання полягає в розподілі на секції великого об’єму даних, для того, щоб запит до даних виконувався в секції тим самим зменшуючи час для сканування. Іноді слід робити акцент на самому моделюванні баз даних, сучасні підходи для NoSql СУБД дещо відрізняються від стандартної реляційної логіки, тому питання продуктивності деколи залежить від логічного проектування [3].

Методи порціювання як технологія горизонтального масштабування допомагає обслуговувати великі таблиці (якщо мова про SQL СУБД) або колекції (мова про NoSQL СУБД) та зменшити загальний час відклику при зчитуванні та загрузці даних для операцій з даними [4].

Розрізняють 2 види порціювання [5]. Першим видом є вертикальне порціювання. Відбувається завдяки поділу атрибутів або колонок даних на декілька різних сутностей чи таблиць. Якщо при вертикальному порціюванні на логічному рівні йде звернення, немов би це 1 таблиця, а на фізичному це декілька, то таке порціювання можна вважати автоматичним. На жаль, автоматичне вертикальне порціювання не підтримується в більшості СУБД.

Другим видом є горизонтальне порціювання. Горизонтальне порціювання відбувається завдяки поділу кортежів або рядків даних на декілька різних сутностей чи таблиць. Наглядним прикладом є таблиця звітів та її порціювання рядків даних за місяцем (див. рис. 1.1). Автоматичне горизонтальне порціювання підтримується більшістю СУБД, та має свої специфіки.

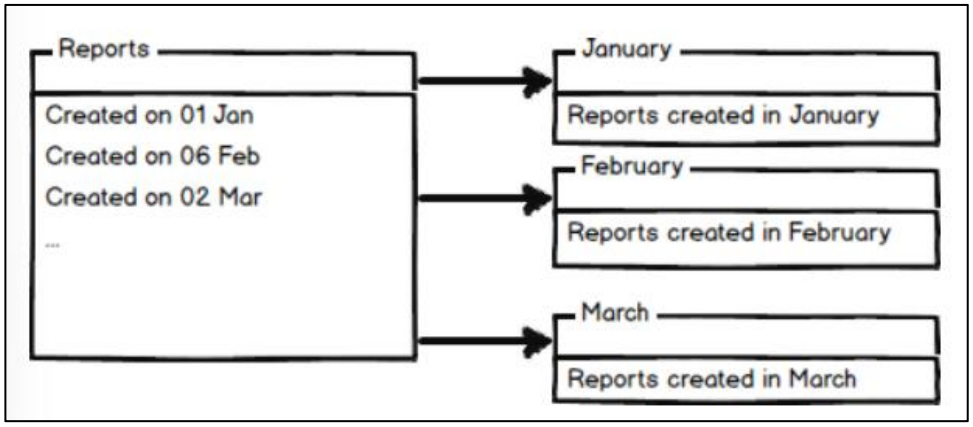

Рисунок 1.1 - Приклад горизонтального порціювання

Але також слід розуміти, що на методи порціювання є певний ряд обмежень, так для більшості SQL-орієнтованих баз даних є обмеження на використання зовнішніх ключів. Інші таблиці не можуть посилатись на таблицю, до якої примінено порціювання, та навпаки. Причиною того є ще одне обмеження на використання головного ключа [6]. За таку оптимізацію необхідно платити. В такому контексті реляційні можливості бази даних дуже обмежені. Тому буде доцільним порівняти методи порціювання, як SQL-орієнтованих систем, так і NoSQL баз даних. Адже NoSQL бази даних також мають методи порціювання, у кожній БД це реалізовано по-своєму. Наприклад, для CosmosDB порціювання задається в 2 етапи [7], перший є логічним порціюванням, та задається користувачем через вказані поля через “ключ порціювання”. Другим етапом є фізичне порціювання, яке задається внутрішніми процесами БД, фізичний сектор може містити 1 або більше логічних секторів. Фізичні сектори можуть бути

задіяні в репліках, оскільки ця база даних є розподіленою, де активно використовуються методи масштабування.

Це наглядно продемонстровано нижче (див. рис. 1.2) на схемі логічних та фізичних секторів.

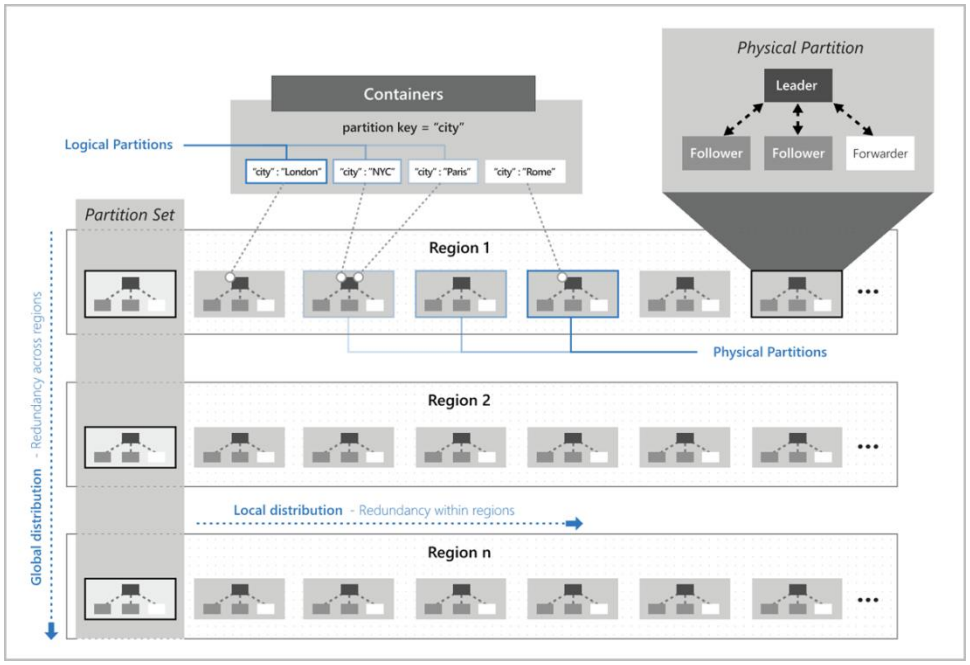

Рисунок 1.2 – Порціювання для CosomosDB

В даному прикладі порціювання відбувається за допомогою простого ключа city. Ключ порціювання може бути складеним та містити більше одного атрибута. Процес розподілу пов’язаний з хешуванням та рівномірним розподілом секцій даних [8]. В області SQL-орієнтованих систем передбачено задання порціювання як створення декількох фізичних файлових просторів та розподіл їх даних за типом порціювання. Типи порціювання включають список, границі, хеш та інші типи. Іншим випадком є задання функції порціювання для можливості повторного використання для декількох таблиць, за функцією порціювання створюється схема порціювання, яка залежить від таблиці, до якої слід використати порціювання, та від функції порціювання, яка буде обрана [9].

Тож вибір СУБД для роботи з великим об’ємом даних та оптимізація обробки операцій над даними є задачею дуже складною. Проте можна

систематизувати набір знань щодо програмних реалізацій СУБД, провести аналіз вимог щодо ключових аспектів якості процесу обробки даних та знайти найбільш підходяще рішення [10].

1.2 Постановка задачі

Проаналізувавши проблемну область, сформулюємо задачу. Задачею є дослідження методів порціювання даних в SQL та NOSQL СУБД, як методів масштабування баз даних. Передбачено розглянути ці методи для найбільш вживаних та популярних систем, порівняти всі ключові аспекти, які саме СУБД найкраще підходять для розробки того чи іншого програмного забезпечення, на що слід звернути увагу при проектуванні програмного забезпечення.

Об’єктами дослідження є різноманітні реляційні та нереляційні бази даних, підходи до порціювання або секціювання даних, особливості їх використання та обмеження, пов’язані з цим.

В ході дослідження необхідно вирішити наступний ряд задач:

– провести аналіз та вибір СУБД для подальшого дослідження методів порціювання;

– провести аналіз реалізації та підходів щодо порціювання в обраних СУБД;

– провести планування експериментального дослідження методів порціювання, що включає розробку критеріїв оцінки якості, розробку схеми БД для предметної області, розробку запитів для дослідження;

– розробити плани порціювання для обраних СУБД та їх реалізувати;

– провести експериментальне дослідження методів порціювання;

– оцінити якість методів порціювання та виробити рекомендації стосовно їх використання.

Дослід включатиме створення баз даних в хмарному середовищі з однаковими умовами щодо кількості реплік кластеру, однаковий розмір дискових просторів, об’єм оперативної пам’яті та обрахункові можливості.

2 ОПИС ПРИЙНЯТИХ ПРОЕКТНИХ РІШЕНЬ

2.1 Аналіз та вибір СУБД для подальшого дослідження методів порціювання

Порціювання даних для різних СУБД є специфічною задачею. Є безліч факторів, що впливають на правильний вибір методу порціювання даних, таких як доступність ресурсів, обсяг даних та чутливість, вимоги бізнесу, тому обраний метод порціювання буде проявляти себе з точки зору ефективності, цей метод буде здійснювати мінімальний вплив на структуру та архітектуру проекту, та буде в рамках бізнес логіки.

З точки зору планування програмного забезпечення передбачено аналіз вимог, узгодження показників якості, шляхи безпеки даних (планування резервного копіювання даних та їх відновлення). При виконанні чітко структурованих кроків стратегії буде досягнуто успіх та можливість уникнути проблеми при виконанні дій у випадковому порядку. Тому при проведенні порціювання даних, слід розуміти що це є шляхом змінення стану персистенції даних та може мати свої наслідки.

Предметна область теж має великий вплив на складність програмного забезпечення. Слід розуміти залежності об’єктів та набір даних, який буде підходити для порціювання. Якщо об’єм даних є малим та відомо, що потенційно ці дані не будуть зростати стрімко в майбутньому, то такий набір даних не потребує порціювання.

Складнощі є у випадку, якщо необхідно втрутитись в існуючу структуру схеми, для NoSQL-орієнтованих систем це зробити дещо краще, адже однією з переваг цих систем є гнучкість щодо даних. Тим не менш в SQL-орієнтованих системах є 2 основні стратегії для проведення порціювання даних, одна з них це створення нової таблиці, що задовільняє умови для порціювання та перенесення даних з існуючої старої таблиці, яка може їх не задовільняти, та видалення старої таблиці. Іншою стратегією є змінення стану існуючої таблиці, щоб відповідати

вимогам порціювання. Всі ці зміни проводяться в рамках міграції як основної одиниці персистентності бази даних.

Для проведення досліджень необхідно обрати максимально релевантні системи управління базами даних. Результати дослідження мають йти в ногу з сучасністю та популярністю систем. Тому оберемо множину альтернатив (СУБД) як найбільш популярні серед користувачів (див. рис. 2.1) згідно даних ресурсу [11].

Worldwide, Nov 2022 compared to a year ago:

Rank	Change	Database	Share	Trend
1		Oracle	27.11 %	-3.5 %
2		MySQL	19.2 %	+2.9 %
3		SQL Server	12.52 %	-2.0 %
4	↑↑	PostgreSQL	6.27 %	+1.7 %
5	↓	Microsoft Access	6.08 %	-1.7 %
6	↓	MongoDB	5.75 %	+0.3 %
7		CosmosDB	4.68 %	+1.5 %
8	↑	Redis	3.01 %	+0.8 %

Рисунок 2.1 – Популярність СУБД

Будемо розглядати задачу вибору СУБД для подальшого дослідження, як задачу багатокритеріального прийняття рішень. Для цього опишемо множину альтернатив та критеріїв вибору.

В якості множини альтернатив розглянемо:

– MySQL;
– SQL Server;
– PostgreSQL;
– MongoDb;

– CosmosDB.

Далі опишемо варіанти, що стосуються критеріїв вибору. В якості множини альтернатив критеріїв вибору розглянемо:

– підтримка порціювання;

– популярність;

– підтримка шардінгу для кластерів;

– кількість записів порцій;

– ціна за годину використання на одній віртуальній машині.

Щодо критерію вибору, то перш за все цікавить підтримка на рівні СУБД порціювання даних, для реляційних СУБД розрізняють порціювання даних таблиці та порціювання даних некластерного індексу, для NoSQL є підримка порціювання для кластерів та відбувається за допомогою логічних та фізичних partition. Але з кожним методом порціювання пов’язані певні види обмежень.

MySQL підтримує горизонтальне порціювання (фізичний поділ рядків таблиці на декілька таблиць та в логічному плані це все ще одна таблиця). Згідно документації [12] серед наявних методів в цій структурі підтримується 6 видів порціювання (див. рис. 2.2).

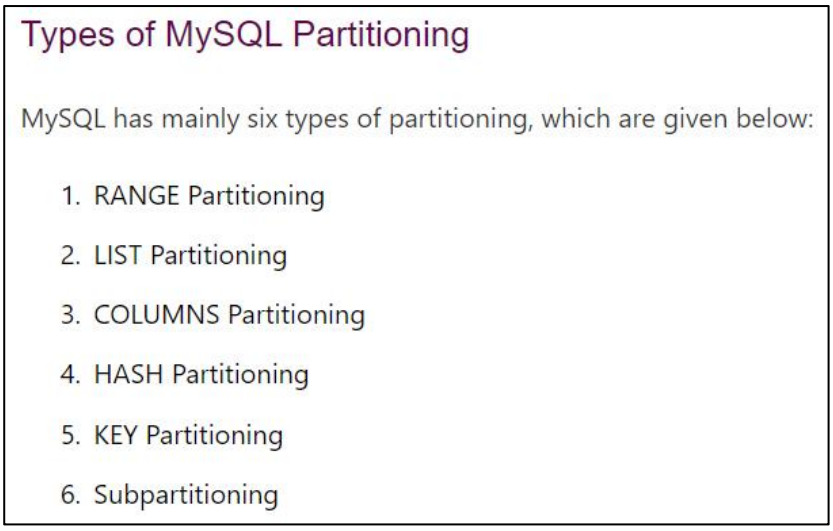

Рисунок 2.2 – Види горизонтального порціювання для MySQL

MySQL не підтримує вертикальне порціювання, а також є обмеження щодо зовнішніх ключів, MySQL порціонує одночасно таблицю та індекси, що з нею пов’язані, що є зручно. Тому за перевищенні вище якості MySQL отримує 1 бал за горизонтальне порціювання, ще 1 за підтримку багатьох видів порціювання, в сумі 2 бали за підтримку порціювання.

MS SQL підтримує горизонтальне порціювання [13], для створення порціювання необхідно створити функцію порціювання, та схему, що є не досить зручно, проте схема може поширюватись на всю базу даних та використовуватись повторно, порціювання в MS SQL не підтримує залежності щодо зовнішніх ключів та головного ключа, за виключенням порціювання за головним ключем. Тому за горизонтальне порціювання 1 бал, за повторне використання для схеми 1 бал, в сумі 2 бали.

PostgreSQL підтримує порціювання так само як і MySQL з різними видами [13], проте ключовою відмінністю є порціювання на рівні ієрархії, на вищому рівні таблиці можно порціювати за типом Range, а далі кожну з порцій розбивати на підпорції за типом List як приклад. Тому за горизонтальне порціювання 1, за використання різних схем порціювання 1, а також за підтримку підпорцій 1, в сумі 3.

MongoDB як NoSQL система підтримує порціювання на рівні чанків [14]. Чанк складається з підмножини сегментованих даних. Кожна частина має нижній і ексклюзивний верхній діапазони на основі ключа фрагмента. MongoDB розділяє чанки, коли вони перевищують налаштований розмір блоку. Як вставки, так і оновлення можуть ініціювати поділ чанків. при зміні розміру чанку потрібно перерозподілити всі записи, що впливає дуже вагомо на час виконання, є внутрішній балансер, який відповідає за рівномірний розподіл даних по чанках про операціях insert, delete. Але чанки орієнтовані на швидкі операції додавання даних, виконання операцій зчитування гальмуються на порядок. За підтриму балансування 1 бал, за порціювання на рівні чанків 1 бал, в сумі 2 бали.

CosmosDB підтримує порціювання за допомогою логічних та фізичних порцій [15], порціювання відбувається вибором ключа для порціювання, ключ може бути 1 та більше атрибутом даних, таким чином користувач може задати стратегію порціювання, це відбувається на логічному блоку, на фізичному підрозбивка порцій оптимізується під фізичний простір пам'яті кластеру. Тому за порціювання логічним рівнем 1 бал, за підтримку фізичного порціювання 1 бал, в сумі 2 бали.

Чим вище показник підтримки порціювання, тим більш релевантнішим вважається СУБД для дослідження.

Показник кількості порцій є максимальною кількістю порцій для даної СУБД. Дані про максимальну кількість порцій було взято з документацій СУБД. Чим більша кількість порцій, тим СУБД більш релевантна для дослідження.

Показник підтримки шардінгу для СУБД є важливим чинником, який дає можливість розбивати дані між різними машинами в середовищі хмарних застосувань. MySQL не підримує шардінг, PostgreSQL підтримує шардінг з обмеженням на ключ по розбивці та з зовнішнім ключем, MS SQL підтримує шардінг на рівні таблиці без залежності зовнішнього ключа, Mongo DB підтримує роботу сховища на кластері з певною кількістю машин, CosmosDB підтримує роботу на кластері з можливістю автоматичного масштабування та зі збільшенням кількості машин при піковому навантаженні.

Тому показники підтримки шардінгу наступні: 0 для MySQL, 2 для PostgreSQL, 3 для MS SQL, 2 для MongoDB, 3 для CosmosDb. Чим більше показник, тим краще.

Ціна за годину використання на одній віртуальній машині – показник для середовища serverless застосувань, дуже критичне для бізнесу, адже чим менше платиш, тим краще.

Тому маємо наступні показники (див. табл. 2.1). Для нормалізації даних по підтримці порціювання, популярності, підтримку шардінгу, максимальна кількість порцій візмемо формулу:

$$a_i = \frac{x_i}{max},$$

де a - нормалізоване значення, max – максимальне значення серед наявної категорії.

Для нормалювання ціни за годину візьмемо теоритично ідеальну ціну d = $0.2, та нормалізуємо за формулою:

$$a_i = \frac{d}{x_i}$$

Таблиця 2.1 – Показники СУБД

	Підтримка порціювання	Популярність	Підтримка шардінгу	Максимальна кількість порцій	Ціна за годину на VM, $
MySQL	2	4	0	8024	0,67
MS SQL	2	4	3	15000	0,51
PostgreSQL	3	3	2	1024	0,63
MongoDB	2	2	2	10000	0,64
CosmosDB	2	2	3	14000	0,32

Задамо наші пріоритети вектором P = {0.25, 0.1, 0.2, 0.2, 0.25}, що задовольняє вимозі $\sum_{j=0}^{m} p_j = 1$ (p_j – пріоритет за критерієм).

Маємо наступну таблицю з нормалізованими даними та даними про пріоритетами (див. табл. 2.2). Слід пригадати оптимізацію за Парето, згідно якої MySQL повністю програє по показникам у порівнянні з MS SQL, а також MongoDB, яка програє по показникам CosmosDB, адже якщо всі показники методу менше або дорівнюють показникам іншого методу, то не має сенсу його використовувати.

Вирішувати дану задачу вибору СУБД будемо за допомогою лінійної адитивної згортки з ваговими коефіцієнтами, за формулою:

$$Z^* = \max \sum_{j=0}^{m} p_i \, a_j,$$

де p – вага пріоритету, a – нормована оцінка критерію.

Таблиця 2.2 – Нормалізовані дані

	Підтримка порціювання	Популярність	Підтримка шардінгу	Максимальна кількість порцій	Ціна за годину на VM, $
MySQL	0.67	1	0	0.53	0.3
MS SQL	0.67	1	1	1	0.39
PostgreSQL	1	0.75	0.67	0.07	0.32
MongoDB	0.67	0.5	0.67	0.67	0.31
CosmosDB	0.67	0.5	1	0.93	0.63
P	0.25	0.1	0.2	0.2	0.25

Результати наведено нижче в порядку від найбільшого (див. табл. 2.3).

Таблиця 2.3 – Результат лінійної адитивної згортки

СУБД	Z*
MS SQL Server	0.765
CosmosDB	0.761
MongoDB	0.563
PostgreSQL	0.553
MySQL	0.4485

Таким чином, було з’ясовано, що найбільш релевантні СУБД для дослідження методів порціювання даних є CosmosDB та MS SQL Server.

Отже, на основі проведення аналізу буде вирішено наступний ряд задач, а саме проведено аналіз вбудованих методів порціювання даних для MS SQL Server та CosmosDB на основі спільної схеми даних та змінної кількості даних, кількості

підключень та навантаження на систему з урахуванням основних величин для заміру.

2.2 Планування експериментальної частини дослідження

Планування експерименту пов’язане з вибором предметної області, а саме з прикладним характером. На основі реальної прикладної предметної області робимо проекцію на ту систему, що найкраще підходить для методів порціювання даних, тобто спрощуємо предметну область для більш наглядного результату. Під отриману загальну схему даних необхідно спроектувати запити та операції над даними для заміру ефективності виконання.

Головними аспектами експериментального дослідження є створення середовищ для MS SQL Server та CosmosDB, що мають однаковий дисковий простір, оперативну пам’ять та обчислювальні можливості, застосування підходів порціювання на цих базах даних на загальній схемі даних та проведення запитів з заміром показників ефективного виконання. Експеримент буде проводитись при різній кількості реплік кластеру та при різному об’єму даних для виявлення умов, при яких та чи інша база даних буде більш ефективною. В якості середовища буде використано віртуальну машину Windows Server від Azure. Машини мають різні характеристики, для експерименту будуть використані машини з 2 ядрами та 4 гб оперативної пам’яті, а також з 4 ядрами та 16 гб оперативної пам’яті. Всі вони будуть ізольовані від побічних процесів, які могли б вплинути на замір ефективності.

2.2.1 Вибір критеріїв з оцінювання якості методів порціювання

Для вимірювання якості методів порціювання потрібні детерміновані показники, що могли би в повній мірі охоплювати різні аспекти якості.

Рівномірність розподілення – важливий критерій порціювання даних. Чим більш рівномірно розподілені дані по сегментам, тим менше буде загальний час для сканування сегменту, а отже й краще оптимізована робота з використання оперативної пам’яті та часу виконання запитів і операцій над даними. Розраховується за формулою

$$P = \frac{\sum (100 - p_i)}{b} * 100,$$

де $b = \frac{100}{n}$; n — кількість сегментів, p_i — частка і-го сегменту.

Відношення часу виконання операцій даних для сегментів. Часто буває так, що запит включає в обробку більше одного сегменту порціювання, а це значить що порціювання може не дати приросту в ефективності виконання. Порціювання даних за відповідністю операцій для сегментів означає залучення якомога меншої кількості порцій даних, в ідеальному випадку 1 операція буде залучати 1 сегмент даних. Метрика вираховується як відношення у відсотках часу виконання запиту з фільтром на сегмент, та час виконання запиту без фільтру.

Важливою характеристикою для даної метрики є Request Unit. Вартість виконання точкового читання (отримання окремого елемента за його ідентифікатором і значенням ключа розділу) для елемента розміром 1 КБ становить одну одиницю запиту або один Request Unit [16]. Відповідно схеми (див. рис. 2.3) для запиту зчитування 1 елементу буде йти 1 RU, для видалення, додавання, оновлення буде задіяно 2 RU, так як дані операції передбачуючь попереднє зчитування. Запит на пошук вже варіюється більшим числом.

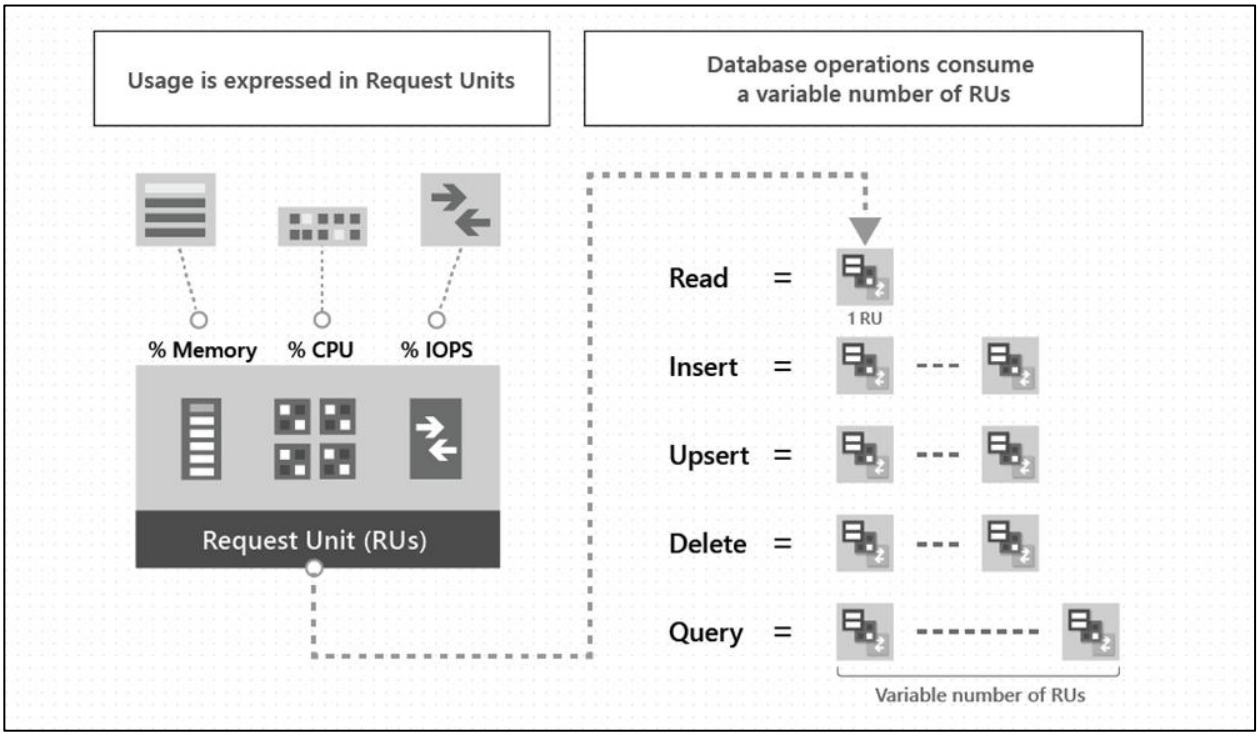

Рисунок 2.3 - Request Unit

Інтеграція в існуючу структуру даних – важливий показник порціювання. Проведення порціювання з мінімальним впливом на персистентність стану даних означає прийнятний рівень інтеграції в існуючу структуру даних. Важливо відмітити що для CosmosDb існує спеціальний програмний інструмент для міграції даних з різних джерел, це може бути формат JSON, CSV файлу, або ж навіть запит з SQL-орієнтованих баз даних [17]. Завдяки цьому застосунку можна впровадити міграцію та замір часу інтеграції в існуючу структуру при різних сценаріях порціювання.

Візуальний інтерфейс можна побачити нижче (див. рис. 2.4)

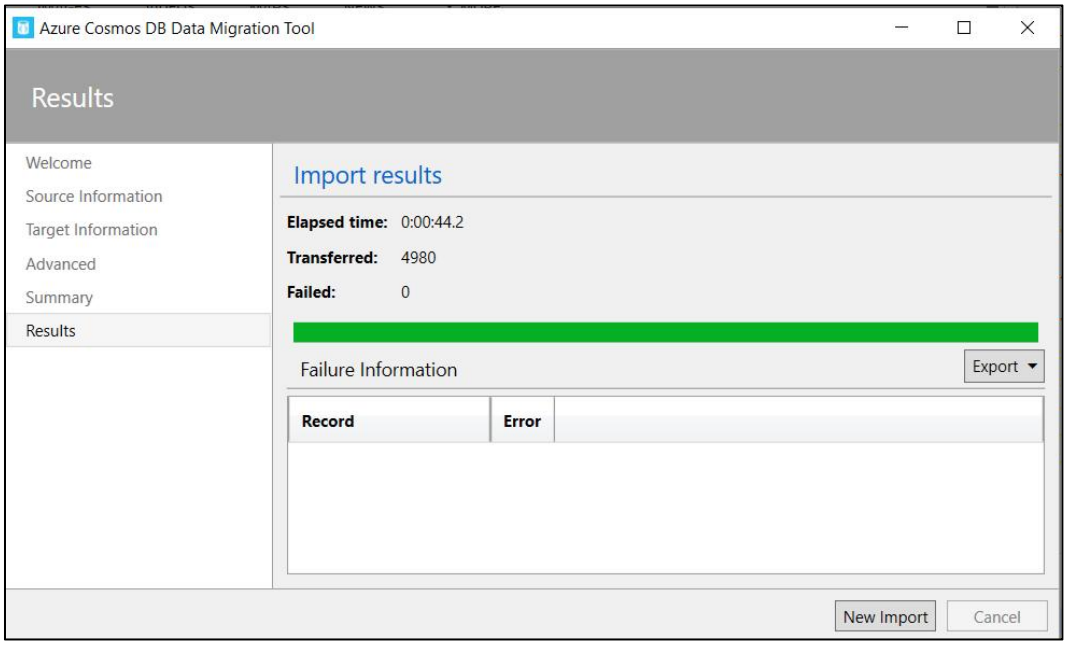

Рисунок 2.4 - Azure Cosmos Db Data Migration tool

Час обчислювальних затрат та час очікування – показник того, наскільки дав приріст виконання операцій над даними при порціюванні. Як відомо, запити до бази даних виконуються пулом потоків та загальний час виконання запиту ділиться на дві частини, одна частина це час обчислювальних витрат (CPU time), час витрачений безпосередньо на виконання запиту. Іншою частиною є час очікування (Wait time). Кожен запит спочатку поміщається в звичайну чергу виконання запитів, при блокуванні ресурсів на виконання транзакції або при великому навантаженні запит не може виконатись в даний квант часу, тому поміщається в чергу очікування (див. рис. 2.5).

Wait Queue має менший пріоритет на виконання, час в якому запит перебуває в черзі очікування або черзі виконання запитів і є wait time. При проведенні порціювання даних очікується зменшення wait time, оскільки порціювання дає можливості розпаралелити виконання запитів та не допустити блокування на рівні ресурсів, також очікується зменшення CPU time при виконанні запиту на секції даних, оскільки менша кількість даних буде задіяна в скануванні.

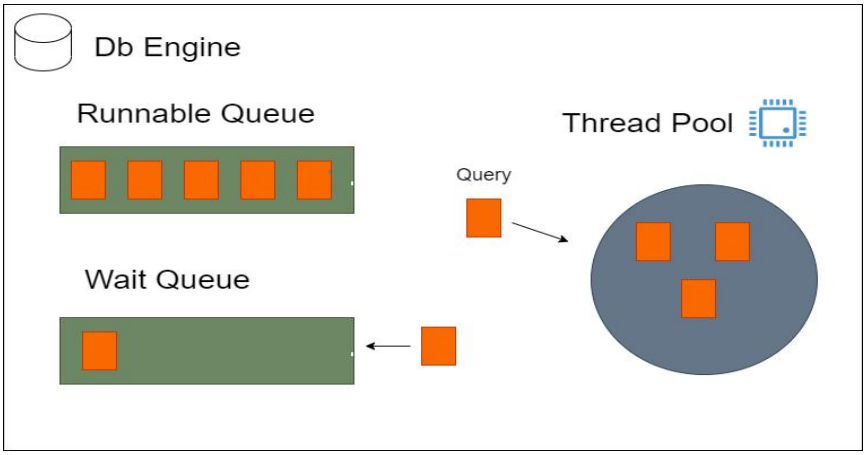

Рисунок 2.5 – Процес обробки запитів

Отже, оцінка якості методів порціювання буде проводитися з урахуванням наступних метрик:

– рівномірність розподілення (%);
– відношення часу виконання операцій даних для сегментів (%);
– час інтеграції в існуючу структуру даних (мс);
– час обчислювальних затрат (мс);
– час очікування (мс).

2.2.2 Аналіз та моделювання предметної області для дослідження

Для розробки структур баз даних для експерименту було обрано область з обрахунками фінансових операцій та формуванням звітності в сфері процесів працевлаштування на базі бюджету держзамовлень. Дана предметна область пов’язана з великими об’ємами даних та високим навантаженням. Процес пов’язаний з працевлаштуванням студентів та заохоченням роботодавців спеціальним видом субсидії, що включає покриття витрат на студентів. На рівні державної структури все має бути чітко та детально, для отримання субсидії роботодавець має задокументувати всі види витрат та підтвердити це чеком або схожим документом. Є окремий поетапний процес підтвердження кожного з

місць працевлаштування для субсидії. З боку студентів це пов’язано з проведенням практики як навчального обов’язкового процесу. Тому ці масові заходи пов’язані з багатьма коледжами та університетами та відбуваються кожного семестру. Формування звітності пов’язано з державним бюджетом, що виділяється щороку та має специфічний процес підтвердження, адже мова йде про мільйони доларів. В рамках дослідження цікавою схемою буде все, що пов’язано з формуванням фінансових звітів, а також другим аспектом, що включає процес формування місця субсидії. Дані можна порціювати на різних рівнях, включаючи семестр як часовий проміжок секції даних, так і регіон для місця реєстрації. Гарантовано, що об’єм великих даних буде рівномірно розподілений, та операції над секціями можуть дати приріст ефективності виконання, ніж сканування всього набору даних.

Проаналізувавши дану предметну область, виділимо основні сутності, що будуть складати загальну схему даних.

Основною сутністю буде subsidized_placement (місце субсидії), воно матиме студента та роботодавця, за ним буде встановлено семестр, під яку програму субсидії підлягає це місце, за якою позицією роботи, кожне місце субсидії має 1 вступ або зарахування студента на позицію.

Програмою субсидії program є сегмент держзамовлення, який має багато підсегментів delivery_program, які встановлюють більш детальні умови держзамовлення.

Зарахування студента enrollment зв’язує деталі студента та семестр term, який підлягає на це місце субсидії.

Семестр term описує часовий проміжок активності місце субсидії.

Навчальний заклад school описує місце навчання студента.

Статус зарахування enrollment_status описує поточний стан зарахування студента, оскільки це контролюється декількома рівнями модераторів, для кожної програми субсидії конфігуруються різні статуси зарахування.

Фінансовий проміжок fiscal_year встановлюватиме рік виділення держбюджету для конкретної програми.

Деталі користувача user_details мають дані всіх користувачів. Там зберігатимуться дані студентів, роботодавців за типом TPH (table per hierarchy).

Проаналізувавши специфіку сутностей маємо загальну діаграму класів, яка відображає основні концепти предметної області (див. рис. 2.6).

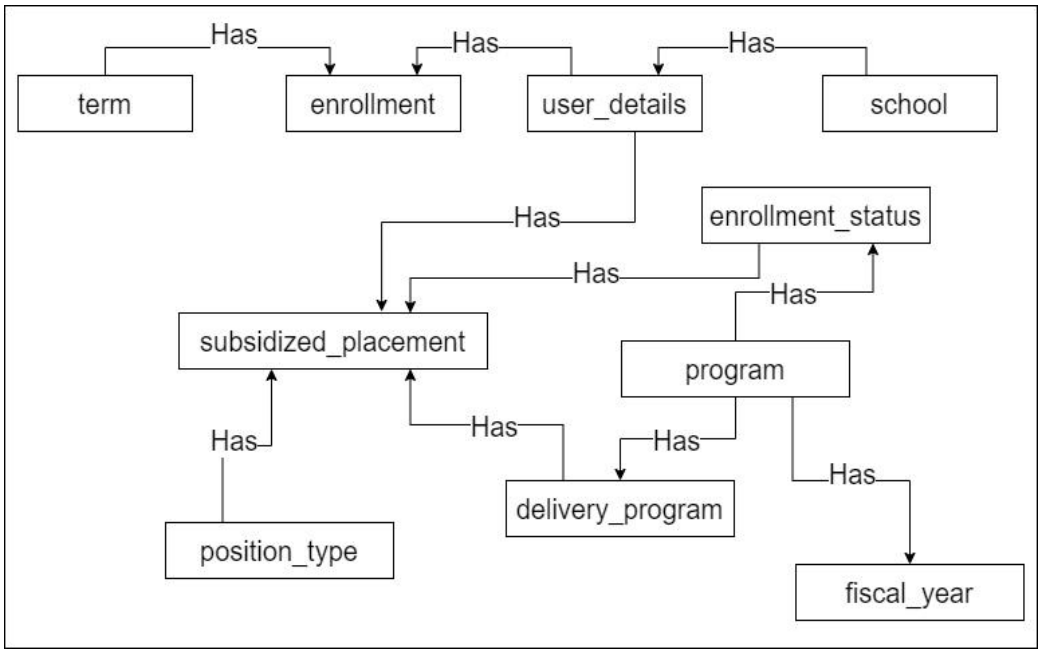

Рисунок 2.6 – Загальна діаграма класів

Проаналізувавши загальну діаграму класів, далі будуть спроектовані логічна схема даних та фізичні моделі БД з урахуванням обмежень, описаних в цьому розділі.

2.2.3 Розробка логічної моделі БД

Проектування логічної схеми БД базується на загальній діаграмі класів. Було розроблено 11 таблиць БД. Зв’язки між сутностями БД (див. рис. 2.7).

Також були спроектовані колонки для сутностей з зазначенням типу даних та чи є цей тип nullable.

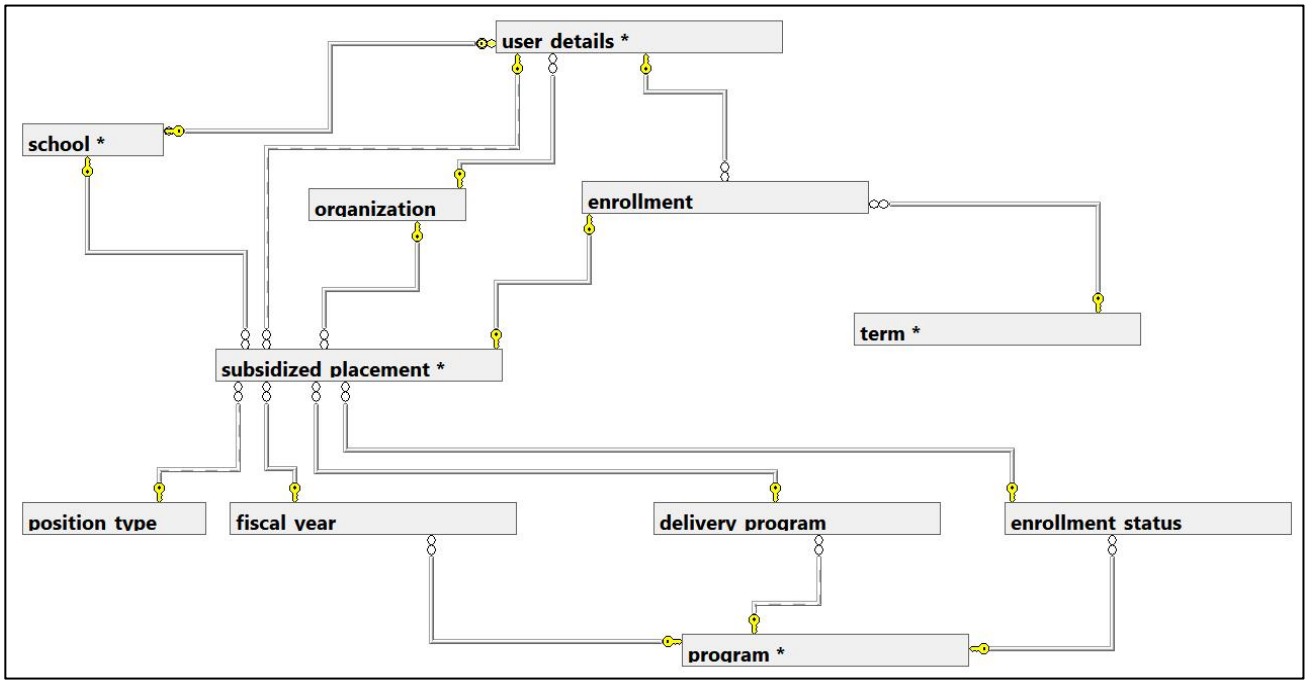

Рисунок 2.7 – Зв’язки сутностей

Далі наведемо характеристики сутностей (див. табл 2.4 - 2.14). Слід зазначити, що колонки для CosmosDB матимуть динамічні типи даних, тому наступні характеристики більш актуальні для MS SQL Server.

Таблиця 2.4 - Сутність fiscal_year (рік фінансування)

Назва	Тип	Пустий за замовчуванням	Опис
id	char(36)	No	Id в форматі Guid
shortId	int	No	Id в числовому форматі
deleted	bit	No	Чи є видаленим
dateCreated	datetimeoffset(7)	Yes	Дата створення
dateUpdated	datetimeoffset(7)	Yes	Дата оновлення
name	nvarchar(255)	Yes	Назва фінансового року
startDate	date	No	Дата старту фінансування
endDate	date	No	Дата закінчення
programId	char(36)	Yes	Ключ на програму
isLocked	bit	No	Чи є закритим
includeInReporting	bit	No	Є відкритим для звітності

Таблиця 2.5 - Сутність subsidized_placement (місце субсидії)

Назва	Тип	Пусто за замовчуванням	Опис
amountPaid	decimal(18, 2)	No	Кількість сплачено
applicationId	nvarchar(255)	yes	Ключ заяви
businessNumber	nvarchar(50)	No	Бізнес номер
city	nvarchar(255)	yes	Місто
companyName	nvarchar(255)	No	Назва компанії
companySize	nvarchar(255)	No	Розмір компанії
country	nvarchar(255)	No	Країна
createdById	char(36)	No	Створено ким
dateCreated	datetimeoffset(7)	No	Створено коли
dateEligibilityAccepted	datetimeoffset(7)	No	Коли прийнято
dateSubmitted	datetimeoffset(7)	yes	Коли відправлено
dateUpdated	datetimeoffset(7)	No	Час оновлення
deleted	bit	No	Чи видалено
deliveryProgramId	char(36)	No	Підпрограма фінансування
editable	bit	No	Чи можна редагувати

Продовження таблиці

Назва	Тип	Пусто за замовчуванням	Опис
enrollmentId	char(36)	No	Ключ на зарахування
EnrollmentStatusId	char(36)	No	Id статус зарахування
fieldOfStudy	nvarchar(255)	yes	Сфера навчання
fiscalQuarter	int	No	Квартал
fiscalYearId	char(36)	No	Ключ на рік фінансування
id	char(36)	No	Id в форматі GUID
jobDescription	nvarchar(MAX)	yes	Описання роботи
jobTitle	nvarchar(255)	yes	Заголовок роботи
legalOrganizationName	nvarchar(MAX)	yes	Ім’я організації
organizationId	char(36)	No	Ключ на організацію
orgType	int	No	Тип організації
paymentStatus	int	No	Статус оплати
phoneNumber	nvarchar(255)	yes	Номер телефону
placementDuration	float	No	Проміжок місця
PlacementDurationInWeeks	int	No	Проміжок в неділях
placementEndDate	date	No	Закінчення місця
placementHourlyWage	decimal(18, 2)	No	Погодинна оплата
placementStartDate	date	No	Старт місця
plannedSubsidyPayment	float	No	Запланова оплата субсидії
positionTypeId	char(36)	Yes	Ключ на тип позиції
postalCode	nvarchar(255)	No	Почтовий індекс
programId	char(36)	No	Ключ на програму
programOfStudy	int	No	Програма навчання
schoolId	char(36)	No	ВУЗ
shortId	int	No	Id в числовому форматі
status	int	No	Статус
studentCity	nvarchar(255)	No	Місто студента
studentConfirmed	bit	No	Підтвердження
studentId	char(36)	No	Ключ на студента
studentNumber	nvarchar(255)	No	Номер залікової книжки
termId	char(36)	No	Ключ на семестр
wageSubsidy	int	No	Рейт субсидії

Таблиця 2.6 - Сутність position_type (тип позиції)

Назва	Тип	Чи є пустим	Опис
id	char(36)	No	Id в форматі GUID
shortId	int	No	Id в числовому форматі
deleted	bit	No	Чи є видалено
dateCreated	datetimeoffset(7)	Yes	Дата створення
dateUpdated	datetimeoffset(7)	Yes	Дата оновлення
tenantId	varchar(8000)	Yes	Ключ на тенанта системи
name	nvarchar(255)	Yes	Назва
paid	bit	No	Чи є оплачуваним
description	nvarchar(MAX)	Yes	Опис
positionTypeOption	int	No	Код позиції

Таблиця 2.7 - Сутність enrollment_status (статус зарахування)

Назва	Тип	Чи є пустим	Опис
id	char(36)	No	Id в форматі GUID
shortId	int	No	Id в числовому форматі
deleted	bit	No	Чи є видалено
dateCreated	datetimeoffset(7)	No	Дата створення
dateUpdated	datetimeoffset(7)	yes	Дата оновлення
name	varchar(255)	yes	Назва
programId	char(36)	No	Програма фінансування
order	int	No	Порядок зарахування

Таблиця 2.8 - Сутність delivery_program (підпрограма фінансування)

Назва	Тип	Чи є пустим	Опис
id	char(36)	No	Id в форматі GUID
name	nvarchar(255)	Yes	Назва
description	nvarchar(MAX)	Yes	Опис
email	nvarchar(254)	Yes	Пошта підпрограми
groupsEnabled	bit	No	Чи доступні ролі в програмі
shortId	int	No	Id в числовому форматі
deleted	bit	No	Чи є видалено
dateCreated	datetimeoffset(7)	Yes	Дата створення
dateUpdated	datetimeoffset(7)	Yes	Дата оновлення
tenantId	varchar(8000)	Yes	Ключ на тенанта системи
programId	char(36)	Yes	Ключ на програму фінансування
allowStudentApplications	bit	No	Чи дозволено студентам заявки
deliveryPartnerId	char(36)	Yes	Ключ на адміністратора

Таблиця 2.9 - Сутність program (програма фінансування)

Назва	Тип	Чи є пустим	Опис
id	char(36)	No	Id в форматі GUID
shortId	int	No	Id в числовому форматі
deleted	bit	No	Чи є видалено
dateCreated	datetimeoffset(7)	Yes	Дата створення
dateUpdated	datetimeoffset(7)	Yes	Дата оновлення
tenantId	varchar(8000)	Yes	Ключ на тенант ситеми
name	nvarchar(255)	Yes	Назва
description	nvarchar(MAX)	Yes	Опис
code	nvarchar(16)	Yes	Код програми фінансування

Таблиця 2.10 - Сутність user_details (дані користувача)

Назва	Тип	Чи є пустим	Опис
firstName	nvarchar(255)	No	Ім’я
lastName	nvarchar(255)	No	Прізвище
middleName	nvarchar(255)	Yes	По батькові
emailAddress	nvarchar(255)	Yes	Пошта
phoneNumber	nvarchar(255)	Yes	Номер телефону
alternateEmail	nvarchar(255)	Yes	Інша пошта
dateCreated	datetimeoffset(7)	Yes	Дата створення
dateUpdated	datetimeoffset(7)	Yes	Дата оновлення
street1	nvarchar(255)	Yes	Адреса 1 рядок
street2	nvarchar(255)	Yes	Адреса 2 рядок
city	nvarchar(255)	Yes	Місто
country	nvarchar(255)	Yes	Країна
username	nvarchar(255)	Yes	нікнейм
deleted	bit	No	Чи є видалено
organizationName	nvarchar(255)	Yes	Ім’я організації
id	char(36)	No	Id в форматі GUID
gender	int	No	Стать
school	char(36)	Yes	ВУЗ
BusinessNumber	nvarchar(50)	Yes	Бізнес номер
shortId	int	No	Id в числовому форматі
yearOfBirth	int	Yes	Рік народження
organizationId	char(36)	Yes	Ключ на організацію
Profession	int	No	Професія

Таблиця 2.11 - Сутність school (ВУЗ)

Назва	Тип	Пустє	Опис
schoolName	nvarchar(255)	Yes	Назва школи
schoolAddressLine1	nvarchar(255)	Yes	Адреса 1 рядок
schoolAddressLine2	nvarchar(255)	Yes	Адреса 2 рядок
schoolAddressCity	nvarchar(255)	Yes	Місто
schoolAddressPostalCode	nvarchar(255)	Yes	Поштовий індекс
schoolAddressCountry	nvarchar(255)	Yes	Країна
contactFirstName	nvarchar(255)	Yes	Контакт ім’я
contactEmail	nvarchar(255)	Yes	Пошта контакту
contactPhoneNumber	nvarchar(255)	Yes	Номер контакту
status	int	No	Статус
emailDomain	nvarchar(MAX)	Yes	Домен
registered	bit	No	Коли зареєстрована
numberOfStudents	int	No	Кількість студентів
deleted	bit	No	Чи є видалено
id	char(36)	No	Id в форматі GUID
administrator	char(36)	Yes	Ключ на адміністратора
dateCreated	datetimeoffset(7)	Yes	Дата створення
dateUpdated	datetimeoffset(7)	Yes	Дата оновлення
shortId	int	No	Id в числовому форматі

Таблиця 2.12 - Сутність term (термін фінансування)

Назва	Тип	Чи є пустим	Опис
id	char(36)	No	Id в форматі GUID
deleted	bit	No	Id в числовому форматі
dateCreated	datetimeoffset(7)	Yes	Дата створення
dateUpdated	datetimeoffset(7)	Yes	Дата оновлення
name	nvarchar(255)	Yes	Назва
startDate	date	No	Старт терміну
endDate	date	No	Закінчення терміну
description	nvarchar(MAX)	Yes	Опис
shortId	int	No	Id в числовому форматі
tenantId	varchar(8000)	Yes	Ключ на тенанта системи

Таблиця 2.13 - Сутність organisation (організація)

Назва	Тип	Чи є пустим	Опис
id	char(36)	No	Id в форматі GUID
shortId	int	No	Id в числовому форматі
deleted	bit	No	Чи є видалено
dateCreated	datetimeoffset(7)	Yes	Дата створення
dateUpdated	datetimeoffset(7)	Yes	Дата оновлення
name	nvarchar(255)	Yes	Назва
orgType	int	Yes	Тип організації
sizeId	char(36)	Yes	Ключ на розмір організації
stateProvinceId	char(36)	Yes	Штат
city	nvarchar(255)	Yes	Місто
address1	nvarchar(255)	Yes	Адреса 1 рядок
address2	nvarchar(255)	Yes	Адреса 2 рядок
postalCode	nvarchar(255)	Yes	Поштовий індекс
website	nvarchar(255)	Yes	Сайт
businessNumber	nvarchar(255)	Yes	Бізнес номер

Таблиця 2.14 - Сутність enrollment (зарахування)

Назва	Тип	Чи є пустим	Опис
id	char(36)	No	Id в форматі GUID
shortId	int	No	Id в числовому форматі
deleted	bit	No	Чи є видалено
dateCreated	datetimeoffset(7)	Yes	Дата створення
dateUpdated	datetimeoffset(7)	Yes	Дата оновлення
tenantId	varchar(8000)	Yes	Ключ на тенант систему
studentId	char(36)	Yes	Ключ студента
termId	char(36)	Yes	Ключ семестру
isComplete	bit	No	Чи є виконано
dateApproved	datetimeoffset(7)	Yes	Дата верифікації
dateDeclined	datetimeoffset(7)	Yes	Дата відмови
status	int	No	Статус

Було наведено всі характеристики сутностей, що стосуються предметної області.

2.2.4 Аналіз атрибутів порціювання для дослідження

Опираючись на предметну область, розглянемо запити, що виникають найчастіше. Таблиця місця субсидії найбільша за розміром, що свідчить про велике навантаження на запис та зчитування даних. Додавання даних відбувається для різних підпрограм системи (tenant), а також кожна підпрограма має свій список вищих навчальних закладів, які використовують систему. Зчитування та агрегування даних необхідне для звітної документації на кожен рік фінансування. Також є підрозбивка звітів щодо семестру. На рівні одної підпрограми додавання місця субсидії відбувається з вказанням програми партнеру фінансування. Для процесу обробки місця субсидії залучені як роботодавці, так й модератори системи, які підтверджують чутливі деталі щодо бюджету тощо.

Проаналізувавши дану область, сформовано пропозицію щодо атрибутів порціювання місця субсидії. Тому потенційно ефективними є атрибути id року фінансування (fiscal_year), id вищого навчального закладу (school), id семестру (term), id програми партнеру фінансування (delivery_program).

Розглянувши більш детально атрибути, було сформовано 2 сценарії для сутності місця субсидії. Сценарій порціювання місця субсидії за вищим навчальним закладом, та порціювання місця субсидії за роком фінансування.

Система включає велику кількість клієнтів та велику кількість ролей. Глобально є адміністратори та модератори, що можуть модерувати підпрограму та внутрішні правила, є роботодавці, які зацікавленні в отриманні субсидій, є студенти, що хочуть працевлаштуватись, а ще є внутрішні відповідальні особи зі сторони вищого навчального закладу, які беруть участь в процесі залучення студентів щодо працевлаштування. Таблиця деталей користувача (user_details) є чутливою та використовується дуже часто в системі через особливості надання

доступу. Користувачі системи студенти додаються в рамках їх навчального закладу. Більш глобально користувачі системи роботодавці (в тому числі й модератори та адміністратори) існують в контексті підпрограми (tenant), як окремої одиниці глобального розгалудження системи.

На основі предметної області сформуємо пропозиції атрибутів порціювання таблиці user_details: id підпрограми системи (tenant), id вищого навчального закладу (school). Але є сумніви щодо рівномірності розбивки користувачів, так як кількість студентів набагато більша, ніж модераторів або роботодавців. Тому як варіант є можливість залучення синтетичного атрибуту, який буде на основі реального. Наприклад, можна взяти за основу день створення користувача та записати день тижня. Як відомо всього існує 7 варіантів, тому рівномірність розбивки даних гарантована.

Розглянувши більш детально атрибути порціювання для сутності деталей користувача, було сформовано 2 сценарії. Розбивка даних деталей користувача за schoolId, розбивка даних деталей користувча за днем тижня його створення.

2.3 Аналіз та розробка планів для методів порціювання для обраних СУБД

2.3.1 Аналіз та розробка планів для методу порціювання для MS SQL Server

Розбиваючи дані, важливо враховувати характеристики даних і типи запитів, які виконуватимуться до них. Ключ порціювання слід вибирати таким чином, щоб він забезпечував оптимальний розподіл даних між вузлами, зменшував обсяг даних, які необхідно передати під час запитів, і мінімізував кількість вузлів, до яких потрібно отримати доступ для отримання необхідних даних (див. рис. 2.8).

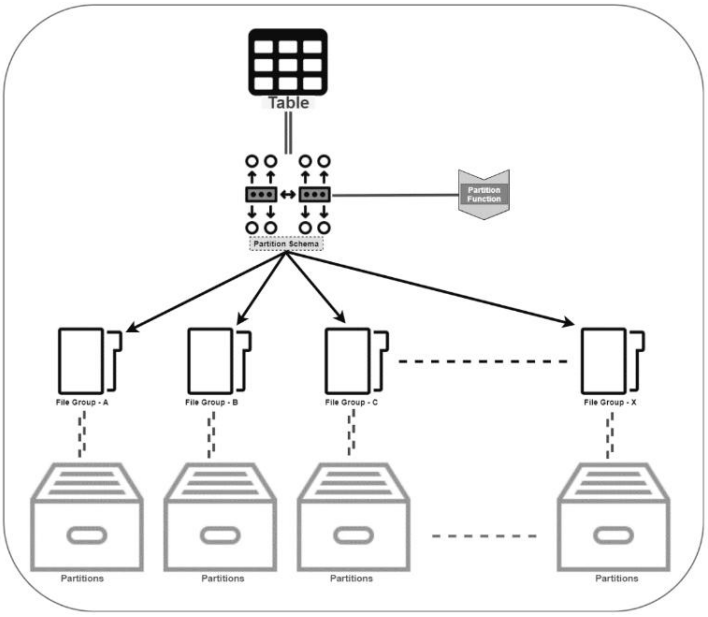

Рисунок 2.8 - Порціювання в MS SQL

Проаналізувавши атрибути для порціювання для дослідження, що були запропоновані в попередньому підрозділі, наведемо концептуальну схему для розбивки даних для деталей користувача (див. рис. 2.9) за підпрограмою системи (tenantId).

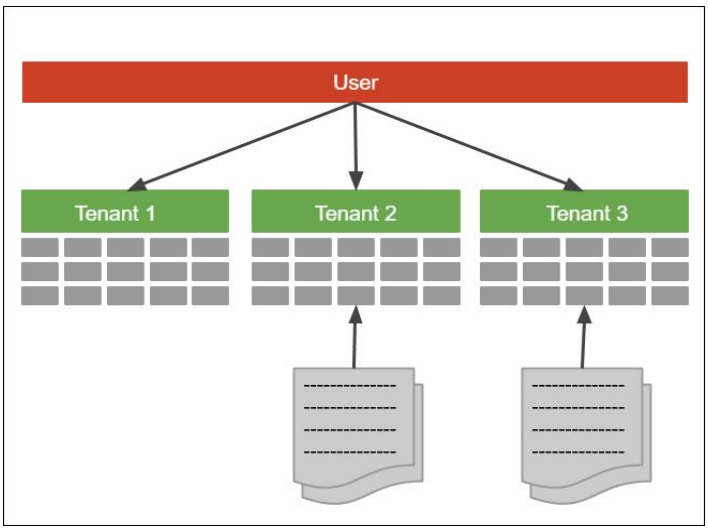

Рисунок 2.9 - Розбивка даних користувача

Для проведення порціювання таблиці в середовищі MS SQL Server необхідно створити функцію порціювання. Для формування звітної документації

беруться до уваги роки фінансування, тому іншими словами кожен рік фінансування абсолютно незалежний для всіх родів запитів щодо місць субсидій (таблиця subsidized_placement).

Тому перелічимо етапи порціювання для MS SQL Server. Спочатку береться до уваги сам процес обробки та розбиття даних, цей об’єкт називається partition function. Далі на основі partition function створюється partition scheme, що відповідає за фізичний розподіл кожного з фрагментів. Коли наявна схема порціювання, то її можна застосувати до таблиці за вказаною колонкою при створенні таблиці. Умовна схема етапів порціювання має наступний вигляд (див. рис. 2.10).

Синтаксис команд наступний:

```
CREATE PARTITION FUNCTION partition_function_name ( input_parameter_type )
AS RANGE [ LEFT | RIGHT ] FOR VALUES ( [ boundary_value [ ,...n ] ] )

CREATE PARTITION SCHEME partition_scheme_name AS PARTITION
partition_function_name TO ( { file_group_name | [ PRIMARY ] })

CREATE TABLE partitionTableName (id int PRIMARY KEY) ON
partition_scheme_name (partition_column_name)
```

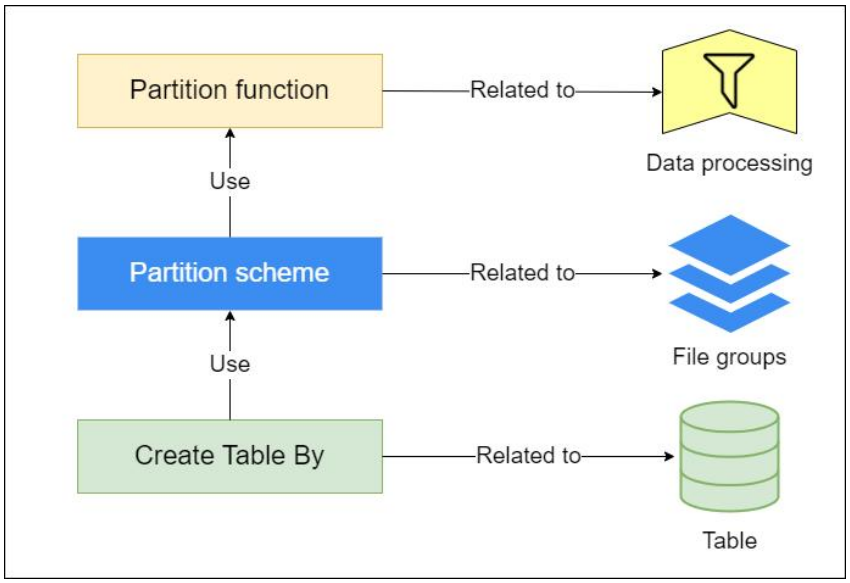

Рисунок 2.10 - Схема етапів порціювання

Така послідовність етапів допомагає при повторному використанні для використання порціювання, як методу горизонтального масштабування, на декількох таблицях та оптимальна для кращого використання фізичного простору.

2.3.2 Аналіз та розробка планів для методу порціювання для CosmosDB

Azure CosmosDB - це глобальна розподілена база даних, яка підтримує кілька API та моделей консистентності. Її можна розглядати як суміш NoSQL-документа та ключ-значення бази даних з реляційними можливостями.

Одна з головних функцій CosmosDB - це порціювання (partitioning), яке дозволяє горизонтально масштабувати базу даних на кілька фізичних серверів. Порціювання дозволяє розбити дані на невеликі частини, які можуть бути розміщені на різних серверах. Це дозволяє збільшити пропускну здатність бази даних та підвищити її доступність шляхом додавання додаткових серверів.

Кожна порція в CosmosDB називається розподілом (partition), і кожен розподіл містить частину даних з бази даних. При додаванні нових даних CosmosDB автоматично розподіляє їх між різними розподілами відповідно до ключа порціювання. Ключ порціювання - це значення, яке використовується для визначення, які дані повинні бути розміщені в одному розподілі.

Ключ порціювання може бути настроєний вручну або вибиратися автоматично. Коли ключ порціювання вибирається автоматично, CosmosDB створює хеш ключа з кожного документа та використовує його для визначення, в якому розподілі документ повинен бути розміщений. Щоб розділити таблицю в CosmosDB, вам потрібно вказати ключ порціювання. Ключ порціювання використовується, щоб визначити, до якого розділу належатиме документ (див. рис. 2.11). Дана схема зроблена з урахуванням аналізу атрибутів порціювання для дослідження. Для прикладу наведено порціювання місця субсидії за id року фінансування.

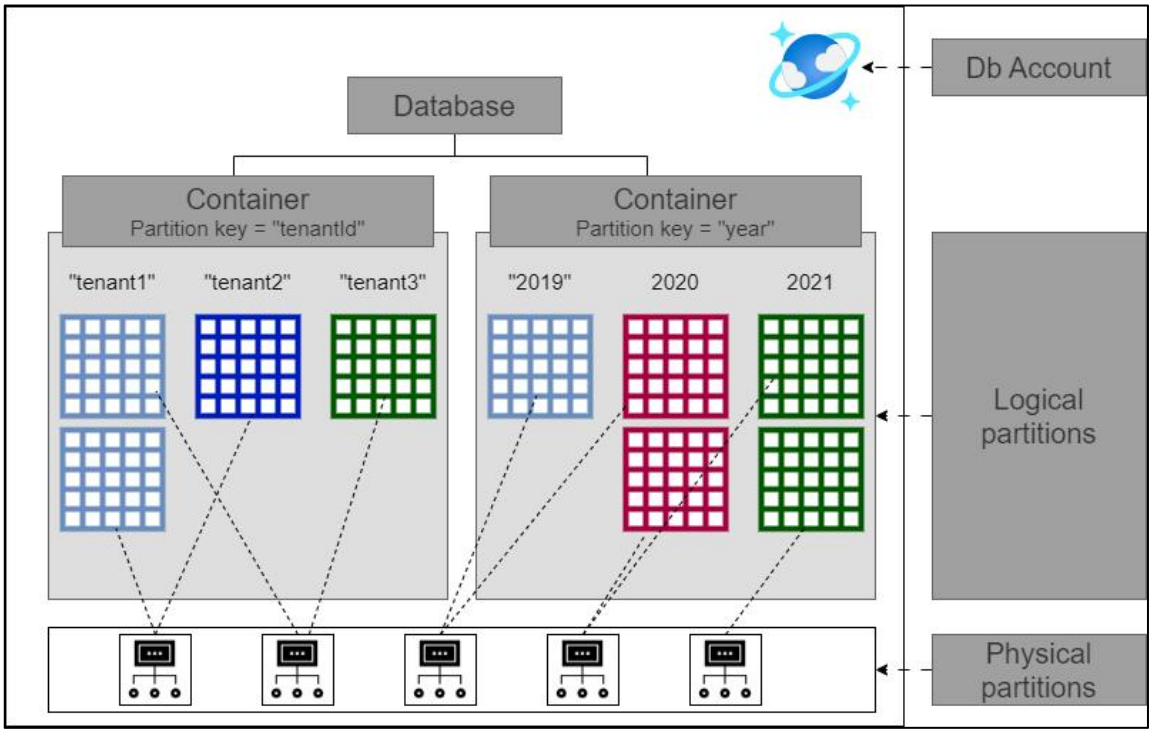

Рисунок 2.11 - Порціювання для CosmosDD

Як видно з схеми вище, порціювання для CosmosDB є високорівневою абстракцією. Оптимальна розбивка даних відбувається на фізичному рівні. Фізичні фрагменти можуть містити 1 або більше логічних фрагментів даних, це залежить від особливостей фізичних характеристик машини, на якій працює CosmosDB.

Логічні фрагменти (англ. logical partitions) - це концепт, що використовується в системі керування базами даних CosmosDB для категоризації та управління даними в розподілених базах даних.

Кожна логічна фрагментація містить окремі документи, які можуть бути розміщені на різних фізичних серверах в рамках різних фізичних розподілів бази даних. Логічні фрагменти дозволяють керувати розподілом даних, що зберігаються в базі даних, та забезпечують можливість масштабування бази даних.

Кожна логічна фрагментація має свій унікальний ідентифікатор, який використовується для доступу до даних, які знаходяться в цьому фрагменті. При створенні бази даних CosmosDB можна визначити кількість логічних фрагментів,

які будуть створені для бази даних. При цьому кількість логічних фрагментів може змінюватись в залежності від потреб користувачів та вимог до бази даних.

Дані, які зберігаються в кожному логічному фрагменті, можуть бути репліковані на різних серверах, що дозволяє забезпечити високу доступність та надійність бази даних. Крім того, розподілення даних між логічними фрагментами дозволяє забезпечити високу продуктивність бази даних, оскільки запити можуть бути оброблені паралельно на різних серверах.

Логічні фрагменти дозволяють розподіляти навантаження між різними серверами та збільшувати пропускну здатність бази даних. Крім того, логічна фрагментація дозволяє забезпечити ефективне використання ресурсів та забезпечити горизонтальне масштабування бази даних.

Для забезпечення ефективності розподілу даних між логічними фрагментами, база даних CosmosDB використовує алгоритми групування даних за значеннями конкретного поля (partition key). Partition key - це ключ, за яким CosmosDB розподіляє дані між різними логічними фрагментами. Цей ключ повинен бути вибраний з урахуванням характеристик даних, що зберігаються в базі даних, і має бути таким, щоб дані з одним значенням partition key зберігались в одному логічному фрагменті.

Якщо partition key вибраний правильно, то можна забезпечити рівномірне розподілення даних між логічними фрагментами та зменшити взаємодію між фрагментами, що зменшує накладні витрати на комунікацію між фрагментами.

Крім того, у CosmosDB є можливість змінювати partition key вже існуючих документів. Це дозволяє змінювати спосіб розподілу даних між логічними фрагментами без необхідності повного перенесення даних.

Узагальнюючи, логічна фрагментація та partition key дозволяють забезпечити горизонтальне масштабування бази даних CosmosDB та забезпечити високу продуктивність та доступність бази даних. При цьому вибір правильного partition key є критичним фактором для ефективності розподілу даних між логічними фрагментами.

Серед сценаріїв порціювання (див. табл. 2.15) для обраної предметної області та структури бази даних було обрано наступні:

- розбивка таблиці user_details по даті створення;
- розбивка таблиці user_details по schoolId ключу;
- розбивка таблиці subsidized_placement по fiscal_year ключу;
- розбивка таблиці subsidized_placement по school ключу.

Таким чином, були розглянуті етапи порціювання для середовища бази даних CosmosDB.

Таблиця 2.15 - Спроектовані схеми порціювання

Спроектовані схеми порціювання	MS SQL Server	CosmosDB
розбивка таблиці user_details	ранжування по значенню типу int дня тижня створення	По хешу дня тижня створення
розбивка таблиці user_details	ранжування по значенню varchar(36) schoolId	по хешу schoolId ключу
розбивка таблиці subsidized_placement	ранжування по значенню типу varchar(36) fiscal_year ключу	по хешу fiscal_year ключу
розбивка таблиці subsidized_placement	ранжування по значенню varchar(36) schoolId	по хешу school ключу

Запропоновані в таблиці 2.15 схеми порціювання будуть використані під час експериментального дослідження.

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

3.1 Розробка фізичної моделі даних

Впровадження програмного забезпечення для кваліфікаційної роботи включало кілька ключових кроків і етапів, включаючи планування, проектування, розробку, тестування та розгортання. У цьому розділі представлено огляд процесу впровадження програмного забезпечення, висвітлюючи найважливіші аспекти та інструменти, які використовуються для дослідження методів порціювання.

Етап розробки був найдовшою та найскладнішою частиною процесу впровадження програмного забезпечення для проведення експерименту. На цьому етапі було використано інформацію, зібрану на етапах планування та проектування, для створення компонентів системи. Цей етап включав написання коду, тестування та налагодження, а також забезпечення того, що компоненти системи працювали разом, як очікувалося. Нижче наведено скрипти створення таблиць.

```
CREATE TABLE fiscal_year (
    id char(36) NOT NULL,
    shortId int NOT NULL,
    deleted bit NOT NULL,
    dateCreated datetimeoffset(7) NULL,
    dateUpdated datetimeoffset(7) NULL,
    name nvarchar(255) NULL,
    startDate date NOT NULL,
    endDate date NOT NULL,
    programId char(36) NULL,
    isLocked bit NOT NULL,
    includeInReporting bit NOT NULL,
    PRIMARY KEY (id)
);

CREATE TABLE subsidized_placement (
    id char(36) NOT NULL,
    amountPaid decimal(18, 2) NOT NULL,
    applicationId nvarchar(255) NULL,
    businessNumber nvarchar(50) NOT NULL,
    city nvarchar(255) NULL,
    companyName nvarchar(255) NOT NULL,
    companySize nvarchar(255) NOT NULL,
    country nvarchar(255) NOT NULL,
    createdById char(36) NOT NULL,
    dateCreated datetimeoffset(7) NOT NULL,
    dateEligibilityAccepted datetimeoffset(7) NOT NULL,
```

```
    dateSubmitted datetimeoffset(7) NULL,
    dateUpdated datetimeoffset(7) NOT NULL,
    deleted bit NOT NULL,
    deliveryProgramId char(36) NOT NULL,
    editable bit NOT NULL,
    enrollmentId char(36) NOT NULL,
    enrollmentStatusId char(36) NOT NULL,
    fieldOfStudy nvarchar(255) NULL,
    fiscalQuarter int NOT NULL,
    fiscalYearId char(36) NOT NULL,
    jobDescription nvarchar(MAX) NULL,
    jobTitle nvarchar(255) NULL,
    legalOrganizationName nvarchar(MAX) NULL,
    organizationId char(36) NOT NULL,
    orgType int NOT NULL,
    paymentStatus int NOT NULL,
    phoneNumber nvarchar(255) NULL,
    placementDuration float NOT NULL,
    placementDurationInWeeks int NOT NULL,
    placementEndDate date NOT NULL,
    placementHourlyWage decimal(18, 2) NOT NULL,
    placementStartDate date NOT NULL,
    plannedSubsidyPayment float NOT NULL,
    positionTypeId char(36) NULL,
    postalCode nvarchar(255) NOT NULL,
    programId char(36) NOT NULL,
    programOfStudy int NOT NULL,
    schoolId char(36) NOT NULL,
    shortId int NOT NULL,
    status int NOT NULL,
    studentCity nvarchar(255) NOT NULL,
    studentConfirmed bit NOT NULL,
    studentId char(36) NOT NULL,
    studentNumber nvarchar(255) NOT NULL,
    termId char(36) NOT NULL,
    wageSubsidy int NOT NULL,
    PRIMARY KEY (id)
);

CREATE TABLE position_type (
    id char(36) NOT NULL,
    shortId int NOT NULL,
    deleted bit NOT NULL,
    dateCreated datetimeoffset(7) NULL,
    dateUpdated datetimeoffset(7) NULL,
    tenantId varchar(8000) NULL,
    name nvarchar(255) NULL,
    paid bit NOT NULL,
    description nvarchar(MAX) NULL,
    positionTypeOption int NOT NULL
}


CREATE TABLE enrollment_status {
    id char(36) NOT NULL,
    shortId int NOT NULL,
```

```
    deleted bit NOT NULL,
    dateCreated datetimeoffset(7) NULL,
    dateUpdated datetimeoffset(7) NULL,
    name nvarchar(255) NULL,
    programId varchar(36) NOT NULL,
    Order int NOT NULL
}

CREATE TABLE delivery_program {
    id char(36) NOT NULL,
    name nvarchar(255) NULL,
    description nvarchar(MAX) NULL,
    email nvarchar(254) NULL,
    groupsEnabled bit NOT NULL,
    shortId int NOT NULL,
    deleted bit NOT NULL,
    dateCreated datetimeoffset(7) NULL,
    dateUpdated datetimeoffset(7) NULL,
    tenantId varchar(8000) NULL,
    programId varchar(36) NOT NULL,
    Order int NOT NULL,
    allowStudentApplications bit NOT NULL,
    deliveryPartnerId char(36) NULL
}

CREATE TABLE program {
    id char(36) NOT NULL,
    shortId int NOT NULL,
    deleted bit NOT NULL,
    dateCreated datetimeoffset(7) NULL,
    dateUpdated datetimeoffset(7) NULL,
    tenantId varchar(8000) NULL,
    name nvarchar(255) NULL,
    description nvarchar(MAX) NULL,
    code nvarchar(16) NULL
}

CREATE TABLE school {
    schoolName nvarchar(255) NULL,
    schoolAddressLine1 nvarchar(255) NULL,
    schoolAddressLine2 nvarchar(255) NULL,
    schoolAddressCity nvarchar(255) NULL,
    schoolAddressPostalCode nvarchar(255) NULL,
    schoolAddressCountry nvarchar(255) NULL,
    contactFirstName nvarchar(255) NULL,
    contactEmail nvarchar(255) NULL,
    contactPhoneNumber nvarchar(255) NULL,
    status int NOT NULL,
    emailDomain nvarchar(MAX) NULL,
    registered bit NOT NULL,
    numberOfStudents int NOT NULL,

    id char(36) NOT NULL,
    shortId int NOT NULL,
    deleted bit NOT NULL,
    dateCreated datetimeoffset(7) NULL,
```

```
    dateUpdated datetimeoffset(7) NULL,
    administrator char(36) NULL
}

CREATE TABLE user_details{
    firstName nvarchar(255) NOT NULL,
    lastName nvarchar(255) NOT NULL,
    middleName nvarchar(255) NOT NULL,
    emailAddress nvarchar(255) NULL,
    phoneNumber nvarchar(255) NULL,
    alternateEmail nvarchar(255) NULL,
    dateCreated datetimeoffset(7) NULL,
    dateUpdated datetimeoffset(7) NULL,
    street1 nvarchar(255) NULL,
    street2 nvarchar(255) NULL,
    city nvarchar(255) NULL,
    country nvarchar(255) NULL,
    username nvarchar(255) NOT NULL,
    deleted bit NOT NULL,
    organisationName nvarchar(255) NULL,
    id char(36) NOT NULL,
    businessNumber nvarchar(50) NULL,
    shortId int NOT NULL,
    yearOfBirth int NULL,
    tenantId char(36) NULL
}

CREATE TABLE organization (
     id char(36) NOT NULL,
     shortId int NOT NULL,
     deleted bit NOT NULL,
     dateCreated datetimeoffset(7) NULL,
     dateUpdated datetimeoffset(7) NULL,
     name nvarchar(255) NULL,
     orgType int NULL,
     sizeId char(36) NULL,
     stateProvinceId char(36) NULL,
     city nvarchar(255) NULL,
     address1 nvarchar(255) NULL,
     address2 nvarchar(255) NULL,
     postalCode nvarchar(255) NULL,
     website nvarchar(255) NULL,
     businessNumber nvarchar(255) NULL,
     PRIMARY KEY (id)
);

CREATE TABLE enrollment (
     id char(36) NOT NULL,
     shortId int NOT NULL,
     deleted bit NOT NULL,
     dateCreated datetimeoffset(7) NULL,
     dateUpdated datetimeoffset(7) NULL,
     tenantId varchar(8000) NULL,
     studentId char(36) NULL,
     termId char(36) NULL,
     isComplete bit NOT NULL,
```

```
      dateApproved datetimeoffset(7) NULL,
      dateDeclined datetimeoffset(7) NULL,
      status int NOT NULL
);

CREATE TABLE term (
      id char(36) NOT NULL,
      shortId int NOT NULL,
      deleted bit NOT NULL,
      dateCreated datetimeoffset(7) NULL,
      dateUpdated datetimeoffset(7) NULL,
      name nvarchar(255) NULL,
      startDate datetimeoffset(7) NOT NULL,
      endDate datetimeoffset(7) NOT NULL,
      description nvarchar(MAX) NULL,
      tenantId varchar(8000) NULL
);
```

Слід зазначити, що даний скрипт є спільною базовою конструкцією як для MS SQL Server так й для CosmosDb SQL, тому тут немає залежностей для foreign key, тому що CosmosDB не підтримує зовнішні ключі [18].

3.2 Розробка скриптів порціювання

Для проведення порціювання таблиці в середовищі MS SQL Server необхідно створити функцію порціювання. Для формування звітної документації беруться до уваги роки фінансування, тому іншими словами кожен рік фінансування абсолютно незалежний для всіх родів запитів щодо місць субсидій (таблиця subsidized_placement). Фізичні файлові простори для років фінансування наступні: 2023, 2022, 2021, 2020, 2019, OTHER. Скрипти для сценаріїв порціювання для MS SQL Server наступні:

```
CREATE PARTITION FUNCTION fiscal_year_partition_func (char(36))
AS RANGE LEFT FOR VALUES (N'2023', N'2022', N'2021', N'2020', N'2019');

CREATE PARTITION SCHEME fiscal_year_partition_scheme
AS PARTITION fiscal_year_partition_func
TO ([2023],[2022],[2021],[2020],[2019], [OTHER]);

ALTER TABLE subsidized_placement
DROP CONSTRAINT PK_subsidized_placement;
ALTER TABLE subsidized_placement
ADD CONSTRAINT PK_subsidized_placement
PRIMARY KEY CLUSTERED (id)
```

```
ON fiscal_year_partition_scheme(fiscalYearId);

CREATE PARTITION FUNCTION tenant_id_partition_func (char(36))
AS RANGE RGHT FOR VALUES ('a5bfdb34-e48a-46c7-a729-b1372b64c843',
'bdce37d5-6c64-4a64-a341-47df629e3cd6', '40283841-4a2e-491e-bf3b-
680c72af43c9', '31f5d954-cc43-44eb-af67-60f4220dc983', '37ac7de0-6eda-4033-
a953-6ef3706addf6');

CREATE PARTITION SCHEME tenant_id_partition_scheme
AS PARTITION tenant_id_partition_func
TO ([tenant1],[tenant2],[tenant3],[tenant4],[tenant5], [OTHER]);

ALTER TABLE user_details
DROP CONSTRAINT PK_user_details;
ALTER TABLE user_details
ADD CONSTRAINT PK_user_details
PRIMARY KEY CLUSTERED (id)
ON tenant_id_partition_scheme(tenantId);
```

Для CosmosDB було обрано сценарій порціювання місця субсидії за роком фінансування, користувача за навчальним закладом, місця субсидії за навчальним закладом та користувача за внутрішньою підсистемою.

Наведемо скрипти для сценаріїв порціювання для середовища бази даних CosmosDB.

Скрипт на порціювання за сценарієм місця субсидії за роком фінансування:

```
subsidized_placement_fiscal_year_partition_1 (fiscalYearId >= 2018)
subsidized_placement_fiscal_year_partition_2 (fiscalYearId >= 2019)
subsidized_placement_fiscal_year_partition_3 (fiscalYearId >= 2020)
subsidized_placement_fiscal_year_partition_4 (fiscalYearId >= 2021)
subsidized_placement_fiscal_year_partition_5 (fiscalYearId >= 2022)
```

Скрипт на порціювання за сценарієм місця субсидії за навчальним закладом:

```
subsidized_placement_school_partition_1 (‘school1’)
subsidized_placement_school_partition_2 (‘school2’)
subsidized_placement_school_partition_3 (‘school3’)
subsidized_placement_school_partition_4 (‘school4’)
subsidized_placement_school_partition_5 (‘school5’)
```

Скрипт на порціювання за сценарієм користувача за навчальним закладом:

```
user_details_school_partition_1 (‘school1’)
user_details_school_partition_2 (‘school2’)
user_details_school_partition_3 (‘school3’)
user_details_school_partition_4 (‘school4’)
user_details_school_partition_5 (‘school5’)
```

Скрипт на порціювання за сценарієм користувача за датою створення:

```
user_details_tenant_partition_1 (1)
user_details_tenant_partition_2 (2)
user_details_tenant_partition_3 (3)
user_details_tenant_partition_4 (4)
user_details_tenant_partition_5 (5)
user_details_tenant_partition_6 (6)
user_details_tenant_partition_7 (7)
```

Таким чином було проведено порціювання таблиці користувачів [19]. Підсумовуючи, розділення даних є важливим аспектом проектування та оптимізації бази даних. Завдяки ефективному розподілу даних базу даних можна зробити більш масштабованою, ефективною та надійною, що має вирішальне значення для великомасштабних програм.

3.3 Розробка запитів

Також важливо описати запити, які будуть виконуватися до бази даних для експерименту. Ці запити повинні бути різноманітними, щоб зробити висновки про ефективність виконання запитів різних типів. Основним типом запитів для вимірювання продуктивності є запити зчитування та запису даних, це доцільно для вимірювання ефективності операцій. Запити опираються на проведений аналіз атрибутів порціювання для дослідження.

Першим таким був запит на читання даних місць субсидії за роком фінансування. Цей запит дозволить виміряти ефективність пошуку індексу в базі даних. А для практичного використання цей запит однозначно корисний, оскільки зазвичай користувачі системи мають можливість працювати з поточним роком фінансування та зосередити весь фокус на ньому. Дамо ім’я запиту “select1”. У SQL цей запит матиме такий вигляд:

```
SELECT id, amountPaid, applicationId, businessNumber, city, companyName, companySize, country, createdById, dateCreated, dateEligibilityAccepted, dateSubmitted, dateUpdated, deleted, deliveryProgramId, editable, enrollmentId, enrollmentStatusId, fiscalQuarter, fiscalYearId,
```

```
jobDescription, jobTitle, legalOrganizationName, organizationId, orgType,
paymentStatus, phoneNumber, placementDuration, placementDurationInWeeks,
placementEndDate, placementHourlyWage, placementStartDate,
plannedSubsidyPayment, positionTypeId, postalCode, programId,
programOfStudy, schoolId, shortId, status, studentId, studentNumber, termId,
wageSubsidy
 FROM subsidized_placement WHERE fiscalYearId = '2023' AND deleted = BIT(0)
ORDER BY dateSubmitted
```

Щоб база даних була масштабованою, дуже важливо ефективно розділити її дані. Це особливо важливо при роботі з великими обсягами даних, які неможливо зберегти на одному вузлі. Розбиття дозволяє розподіляти дані між кількома вузлами, що може покращити продуктивність операцій читання та запису, а також підвищити загальну надійність і доступність бази даних. Дамо ім'я запиту "update1". Розробимо запит для записування даних з фільтром на секцію порціювання:

```
UPDATE subsidized_placement SET wageSubsidy = 3000
WHERE fiscal_year = 2021 AND id = 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
```

Далі розробимо запит для створення subsidized_placement. Дамо ім'я запиту "insert1":

```
INSERT INTO subsidized_placement
([amountPaid],[applicationId],[businessNumber],[city],[companyName],[countr
y],[createdById],[dateCreated],[dateEligibilityAccepted],[dateSubmitted],[d
ateUpdated],[deleted],[deliveryProgramId],[editable],[enrollmentId],[Enroll
mentStatusId],[fieldOfStudy],[fiscalQuarter],[fiscalYearId],[id],[jobDescri
ption],[jobTitle],[legalOrganizationName],[organizationId],[orgType],[payme
ntStatus],[phoneNumber],[placementDuration],[PlacementDurationInWeeks],[pla
cementEndDate],[placementHourlyWage],[placementStartDate],[positionTypeId],
[postalCode],[programId],[programOfStudy],[schoolId],[shortId],[studentCity]
,[studentConfirmed],[studentId],[studentNumber],[termId],[wageSubsidy])
VALUES
(<amountPaid>,<applicationId>,<businessNumber>,<city>,<companyName>,<compan
ySize>,<country>,<createdById>,<dateCreated>,<dateEligibilityAccepted>,<dat
eSubmitted>,<dateUpdated>,<deleted>,<deliveryProgramId>,<editable>,<enrollm
entId>,
<EnrollmentStatusId>,<fieldOfStudy>,<fiscalQuarter>,<fiscalYearId>,
<id>,<jobDescription>,<jobTitle>,<legalOrganizationName>,<organizationId>,<
orgType>,<paymentStatus>,<phoneNumber>,<placementDuration>,<PlacementDurati
onInWeeks>,<placementEndDate>,<placementHourlyWage>,<placementStartDate>,<p
lannedSubsidyPayment>,<positionTypeId>,<postalCode>,<programId>,<programOfS
tudy>,<schoolId>,<shortId>,<status>,<studentCity>,<studentConfirmed>,<stude
ntId>,<studentNumber>,<termId>,<wageSubsidy>)
```

Запит для видалення перевіряти недоцільно, так як в системі реалізована технологія “Soft delete”, кожен з сутностей має поле deleted, яке описує чи видалений він. Технічно запит видалення буде ще одним запитом Update:

```
UPDATE subsidized_placement set deleted = 1 WHERE id = 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
```

Додамо запит на агрегування даних “count1”:

```
SELECT COUNT(*) FROM subsidized_placement WHERE fiscalYearId = 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx';
SELECT COUNT(*) FROM subsidized_placement WHERE deliveryProgramId = 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx';
```

Далі був розроблений запит для отримання деталей користувача з додатковим фільтром на tenantId. Дамо ім’я запиту “select2” :

```
SELECT * FROM user_details WHERE tenantId = 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx' AND emailAddress = 'example@email.com';
```

Додамо запит з прикладом агрегування даних “count2”:

```
SELECT COUNT(*) FROM user_details WHERE tenantId = 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx';
SELECT COUNT(*) FROM user_details WHERE  weekDayCreated = 1;
```

Далі запит на оновлення “update2”:

```
UPDATE user_details set firstName = <name>, secondName = <secondName> WHERE tenantId = 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
```

Далі запит на додавання “insert2”:

```
INSERT INTO user_details ([id], [username], [firstName], [secondName], [tenantId]) VALUES (<id>, <username>, <firstName>, <secondName>, <tenantId>);
```

Слід зазначити, що запити з використанням Join не будуть розглядатись в рамках даного дослідження. Cosmos DB має ряд обмежень щодо використання оператору Join. Ця проблема вирішується моделюванням нової схеми, що виключає з'єднання сутностей між собою, всі чутливі сутності для Join поміщаються всередину одразу як внутрішній об'єкт у випадку зв'язку один до багатьох. Якщо потрібно залучити зв'язок багато до багатьох, то варіантом вирішення є зберігання масиву зовнішніх ключів для двох сутностей.

Підсумовуючи, з розробкою сценаріїв розділення таблиць ми маємо базові дані, що необхідні для проведення експерименту з операціями читання та запису даних. Тепер ми можемо перевірити ефективність розділення таблиць у покращенні продуктивності запитів і керування даними. За допомогою відповідних запитів SQL, таких як запити на фільтрування, групування, об'єднання, оновлення, ми можемо побачити, як розділення таблиці може призвести до швидшого часу відповіді на запит і кращого використання системних ресурсів.

Важливо відзначити, що розділення таблиці є лише одним із кількох методів, які можна використовувати для оптимізації продуктивності бази даних, і його ефективність може змінюватися залежно від розміру та складності даних, якими керують. Тому важливо ретельно оцінити використання розбиття таблиці [20] в кожному конкретному випадку, щоб визначити, чи буде воно корисним.

Таким чином, розробка сценаріїв порціювання таблиць є важливим кроком у процесі оптимізації продуктивності бази даних. Це дозволяє експериментувати з продуктивністю операцій читання та запису даних у розділеній таблиці та оцінювати ефективність розділення таблиці для покращення часу відповіді на запити та використання системних ресурсів.

3.4 Опис програмного середовища з проведення експериментів

Для проведення експериментів було обрано віртуальну машину на ОС Windows 11 з 8 гб оперативної пам’яті та 64-бітною архітектурою. Ресурс надається системою Azure на основі студентської підписки. Характеристики машини наведено нижче (див. рис. 3.1 - 3.2).

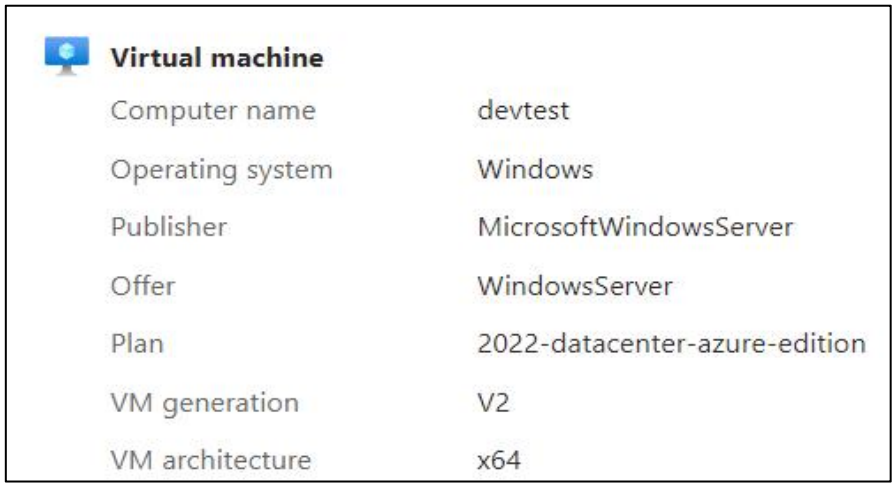

Рисунок 3.1 - Опис віртуальної машини

Рисунок 3.2 - Характеристики віртуальної машини

Для під’єднання до віртуальної машини (див. рис. 3.3) використовується протокол RDP (Remote Desktop Connection).

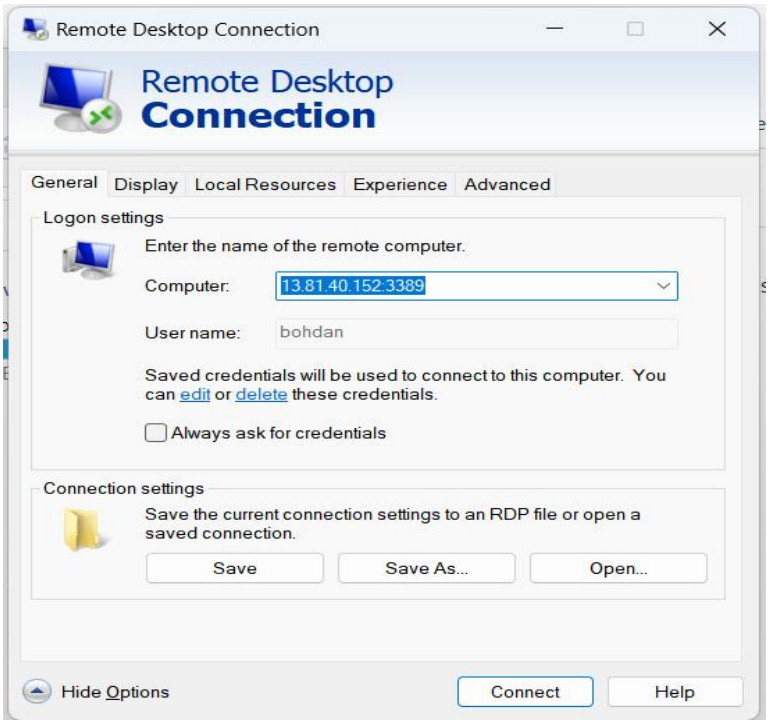

Рисунок 3.3 - Remote Desktop Connection

Для системи Windows є влаштоване програмне забезпечення, що підтримує цей протокол (див. рис. 3.4).

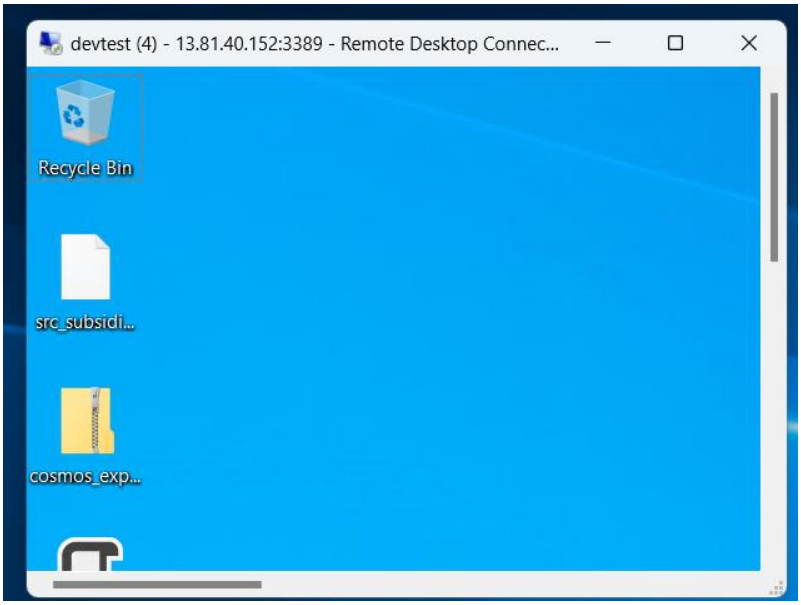

Рисунок 3.4 - Програмне забезпечення RDP

Для роботи з Cosmos DB використовувався веб застосунок (див. рис. 3.5). Версія API Cosmos DB: 2018-12-31. SDK: Microsoft.Azure.Cosmos 3.32.3

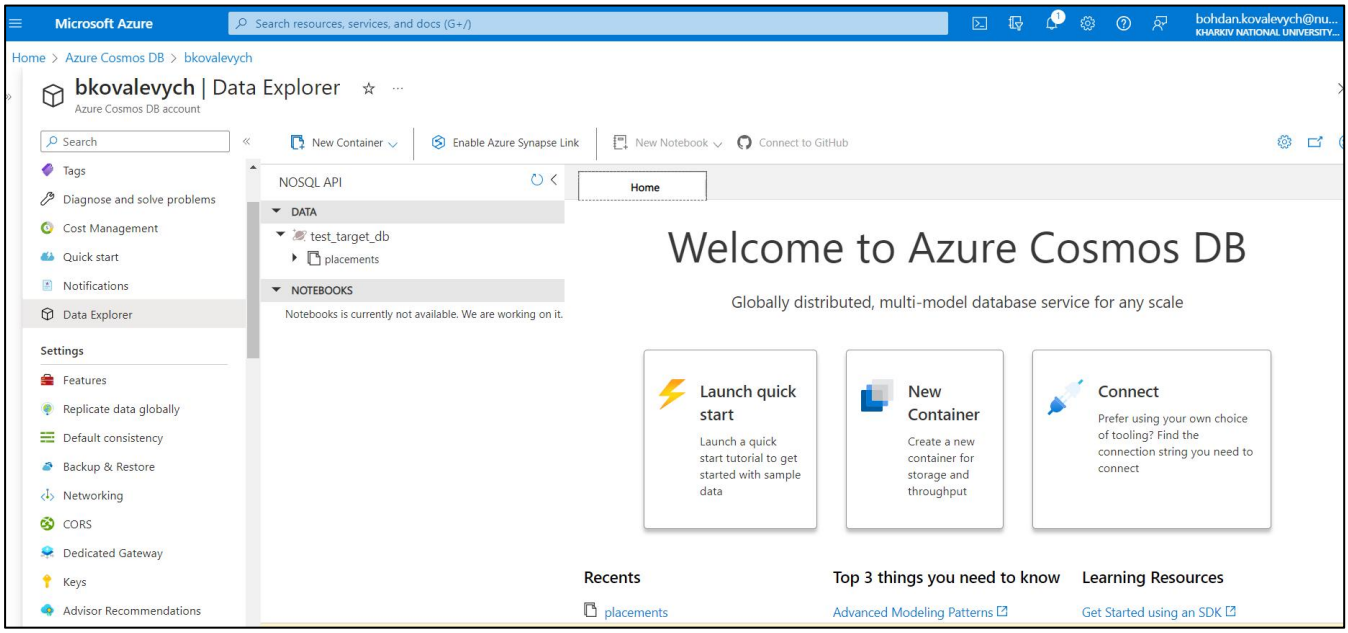

Рисунок 3.5 - Веб застосунок для роботи з CosmosDb

Для роботи з MS SQL Server використовувалась SQL Server Management Studio 19.02 (див. рис. 3.6). Версія MS SQL Server: 2019 (Compatibility level 150). SDKs: Microsoft.SqlServer.DacFx 161.8089.0, System.Data.SqlClient 4.8.5.

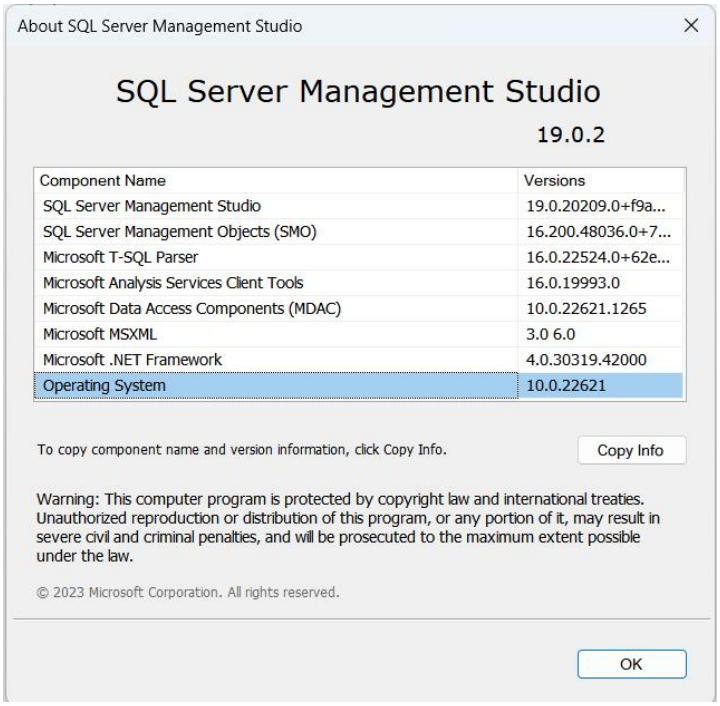

Рисунок 3.6 - SQL Management Studio

Для розробки програмного застосунку, який виконував методи порціювання та необхідні заміри, використовувалась Visual Studio 2022. Тип програми: Console App на основі .NET 6, C# 10.

4 ПРОВЕДЕННЯ ЕКСПЕРИМЕНТАЛЬНОГО ДОСЛІДЖЕННЯ ПОРЦІЮВАННЯ

4.1 Результати експериментальних досліджень

4.1.1 Дослідження відносно метрики рівномірності розподілу даних порціювання

Виділимо цікаві аспекти для метрики рівномірності розподілу порціювання. Акцент робиться не на кількості сегментів або їх об’єму, а саме рівномірності їх розподілу. Роздивимось сценарій порціювання місця субсидії за id року фінансування. Для MS SQL Server після задання FileGroup для кожного значення id року фінансування, а також файлів всередині бази даних відповідно до кожного FileGroup було створено функцію порціювання, а також схему порціювання. Після задіяння методу порціювання бачимо наступний результат (див. рис. 4.1). 3 роки займають найбільше об’єму у відсотках, порціювання за даним сценарієм вийшло не рівномірним (див. табл. 4.1). В свою чергу CosmosDB використовує хеш від значень року фінансування, тому за внутрішньою реалізацією було створено 3 сегменти (див. рис. 4.2), які краще за рівномірністю розподілу.

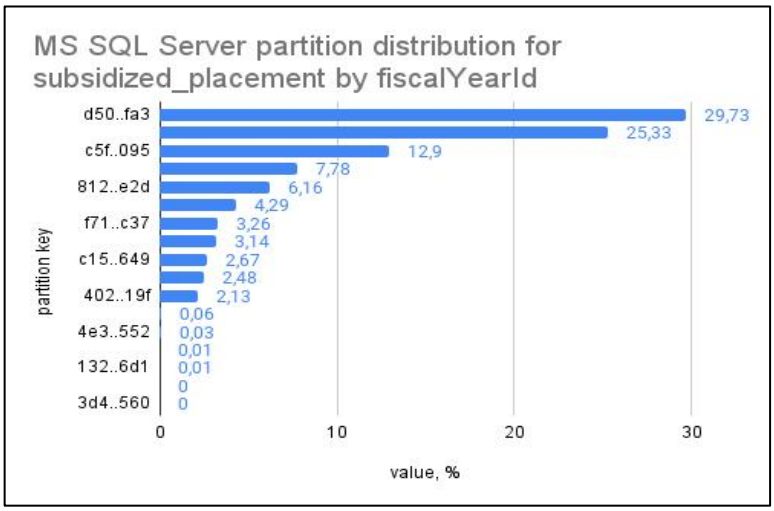

Рисунок 4.1 - Порціювання для MS SQL Server для сценарію з місцем субсидії

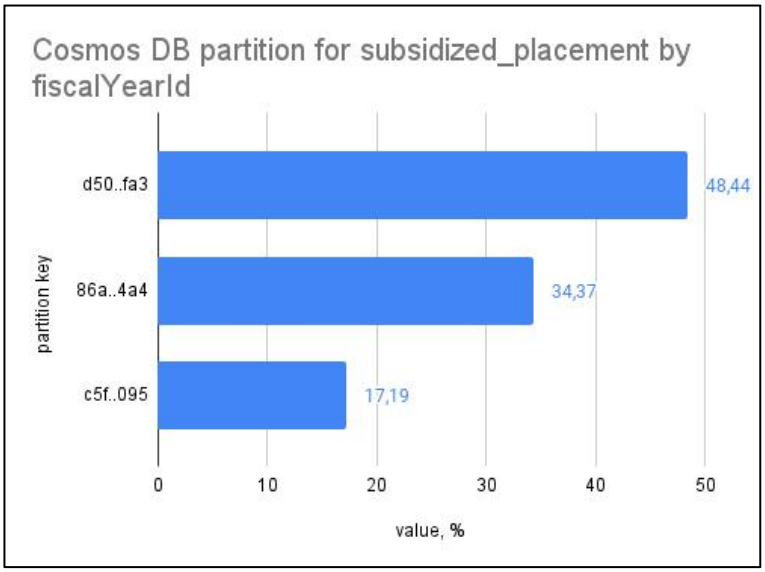

Рисунок 4.2 - Порціювання для Cosmos DB для сценарію з місцем субсидії

Тому для вирішення проблеми рівномірності розподілу, розглянемо приклад з використанням синтетичного ключа для двох СУБД. Тому задамо сценарій порціювання даних користувача за днем тижня його створення. Всього буде розповсюджено 7 сегментів, по одному для кожного дня. Як можна бачити (див. рис. 4.3 - 4.4) сценарій з синтетичним ключем порціювання показав себе краще за відсотком відхилення для MS SQL Server.

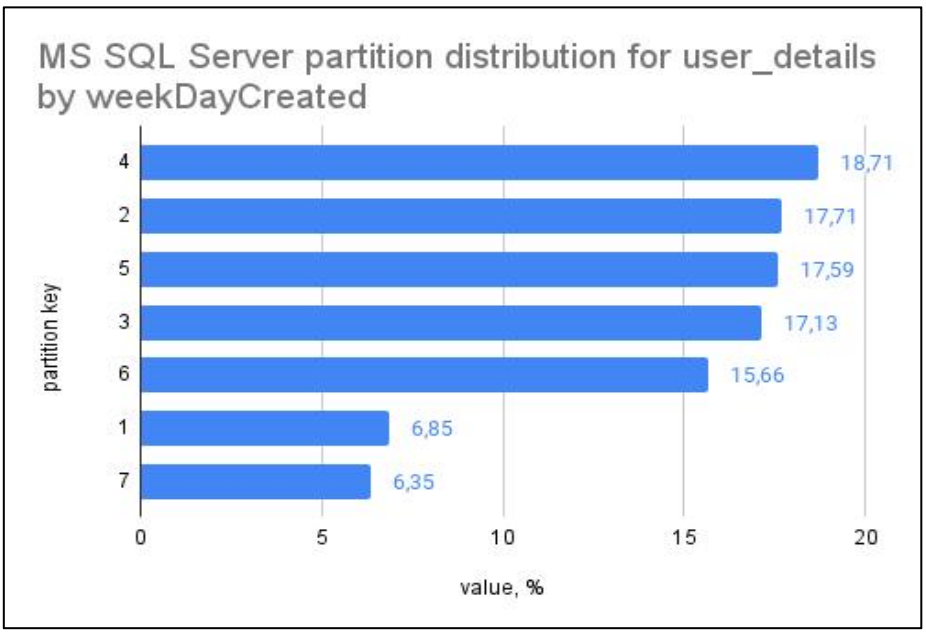

Рисунок 4.3 - Порціювання для MS SQL Server для сценарію з користувачем

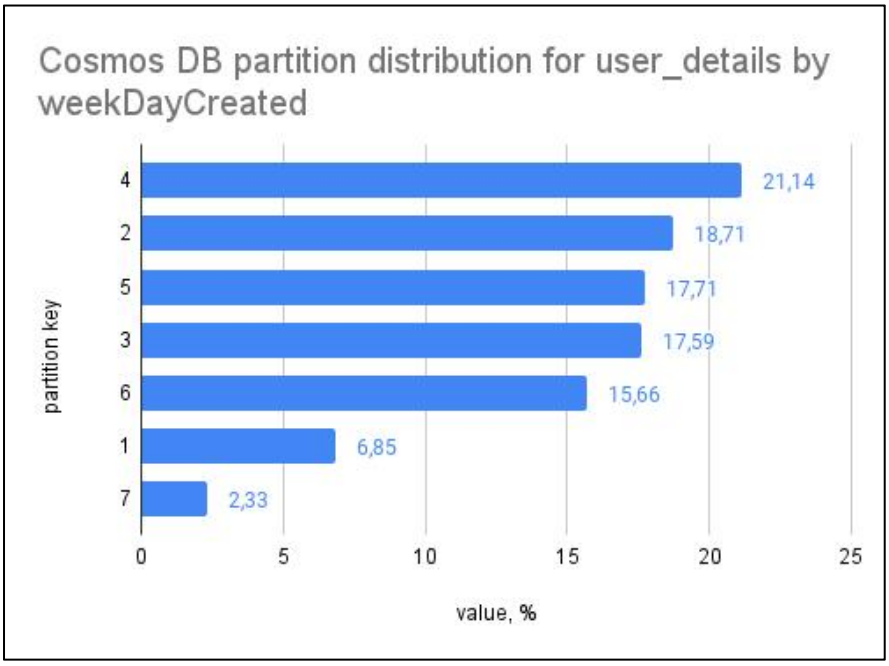

Рисунок 4.4 - Порціювання для Cosmos DB для сценарію з користувачем

Нижче можна побачити результати проведеного експерименту за метрикою рівномірності розподілу (див. табл. 4.1), більш детально було описано метрику в підрозділі 2.2.1.

Таблиця 4.1 - Результати рівномірності порціювання для сценаріїв

Сутність	Сценарій порціювання	MS SQL	CosmosDB
Місце субсидії	за роком фінансування	31.73%	70.07%
	за вищим навч. закладом	25.2%	60%
Дані користувача	за вищим навч. закладом	23%	65%
	за датою створення	72.17%	65.89%

Отже було проаналізовано результати експерименту задіяних сценаріїв порціювання з перевіркою на метрику рівномірності розподілу сегментів. Виявлено найбільш ефективний розподіл з застосуванням синтетичного ключа (день тижня створення користувача).

4.1.2 Дослідження відносно метрики відношення часу виконання операцій для сегментів даних порціювання

Розглянемо дослідження з точки зору відношення часу виконання операцій для сегментів порціювання. Є два крайніх випадки щодо сегментів операцій, якщо запит в обробці залучає всього 1 сегмент, то це відповідає 100% відповідності операцій. Якщо ж запит залучив усі сегменти, то це 0% відповідності операцій. Ця метрика пов’язана з рівномірністю розподілу порціювання, де потрібно знайти баланс. Метод порціювання вважається ефективноим при рівномірному розподілу даних та при операціях, які будуть залучати мінімальну кількість сегментів. Відповідно чим менше буде задіяно ресурсів для виконання запиту, тим більше відношення часу виконання.

Тож перейдемо до розгляду сценаріїв порціювання та розроблених запитів. Зазначимо, що розроблені запити ефективно підійдуть для сценаріїв зі справжніми ключами порціювання і по великому рахунку це залежить від бізнес логіки та самої структури даних, а не від бази даних чи методу порціювання. Проте більш цікавим є сценарій з синтетичним ключем, адже він показав високу ефективність в плані рівномірності розподілу даних, тому даний етап дослідження буде стосуватись тільки сценарію розподілу за синтетичним ключем, а саме даних користувача за днем тижня створення.

Для дослідження в середовищі Cosmos DB виконали запити “select1”, “insert1”, “update1” для user_details спочатку з накладенням фільтру на синтетичний ключ, а потім виконали запити без фільтру на синтетичний ключ. Кожен з запитів має свій Request charge, що вимірюється в Request Unit. Щоб була можливість порівняти з MS SQL Server, було взято відсоткове відношення request charge для кожного з запитів (див. табл. 4.2).

Таблиця 4.2 - Відповідність операцій для CosmosDB

Назва запиту	З фільтром, RU	Без фільтру, RU	Відношення, %
select1	22,7	24,2	93,8
update1	18,2	25,3	71,94
insert1	33,12	35,5	93,3

Для дослідження в середовищі MS SQL Server виконали ідентичні запити з накладенням фільтру на синтетичний ключ та без нього, проте замір довелось робити відносно часу виконання (elapsed time, ms). Було отримано наступні результати (див. табл. 4.3) для MS SQL.

Таблиця 4.3 - Відповідність операцій для MS SQL

Назва запиту	З фільтром, ms	Без фільтру, ms	Відношення, %
select1	9,5	17,2	71,97
update1	13,1	23,3	67,88
insert1	22	28	78,57

Для порівняння результатів було взято саме колонку відношення (%). Дані відношення можна зіставити в наступній діаграмі (див. рис. 4.5).

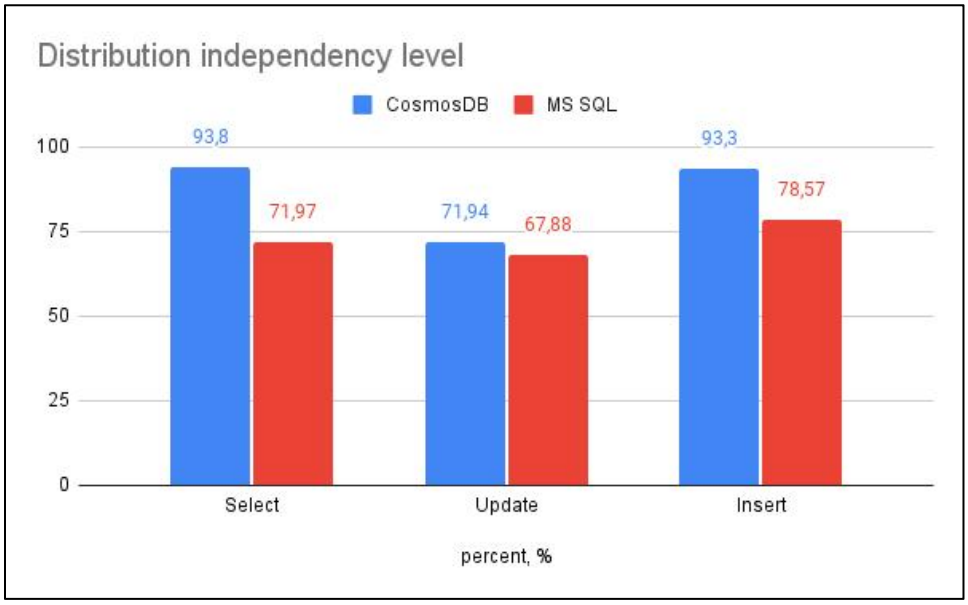

Рисунок 4.5 - Відповідність операцій для сегментів даних порціювання

Отже було проаналізовано результати експерименту задіяних сценаріїв порціювання з перевіркою на метрику відношення часу виконання операцій для сегментів порціювання. Відношення часу для Cosmos Db показує більший приріст ефективності.

4.1.3 Дослідження відносно метрики часу інтеграції в існуючу структуру даних

Перейдемо до метрики інтеграції в існуючу структуру даних. Для кожного сценарію було створено нову базу даних, потім підготовлено контейнер з відповідним ключем порціювання для Cosmos DB та таблицю для MS SQL Server з fileGroups, partition function, partitition function. Далі на основі існуючої структури даних було виконано міграцію даних. Спочатку міграція відбувалась для малого розміру даних (120 Мб), потім середнього (600Мб), потім повного розміру (1200 Мб) наявних таблиць. Кожен зі сценаріїв був заміряний в мілісекундах. Сценарії наступні. Місце субсидії за програмою фінансування — subsidized placement by delivery program id. Місце субсидії за роком фінансування — subsidized placement by term. Деталі користувача за підпрограмою — user details by tenant id. Деталі користувача за днем тижня створення — user details by date created. Результати замірів ефективності наведені нижче (див. рис. 4.6 - 4.8).

Для всіх розмірів таблиць було виявлено ефективність виконання з боку MS SQL, проте слід пам’ятати, що технічно для Cosmos Db достатньо задати ключ та назву контейнера, хоч і повільно, але міграція буде виконана. Що ж до MS SQL Server, то необхідно виконати більш складний скрипт й існує більша ймовірність помилок, ніж для Cosmos DB.

Також була виявлена особливість для сценаріїв, База даних Cosmos DB більш стабільна в міграції даних при збільшенні кількості значень ключа порціювання. Що ж до MS SQL, то при збільшенні значень ключа порціювання існує тенденція сповільнення часу для міграції.

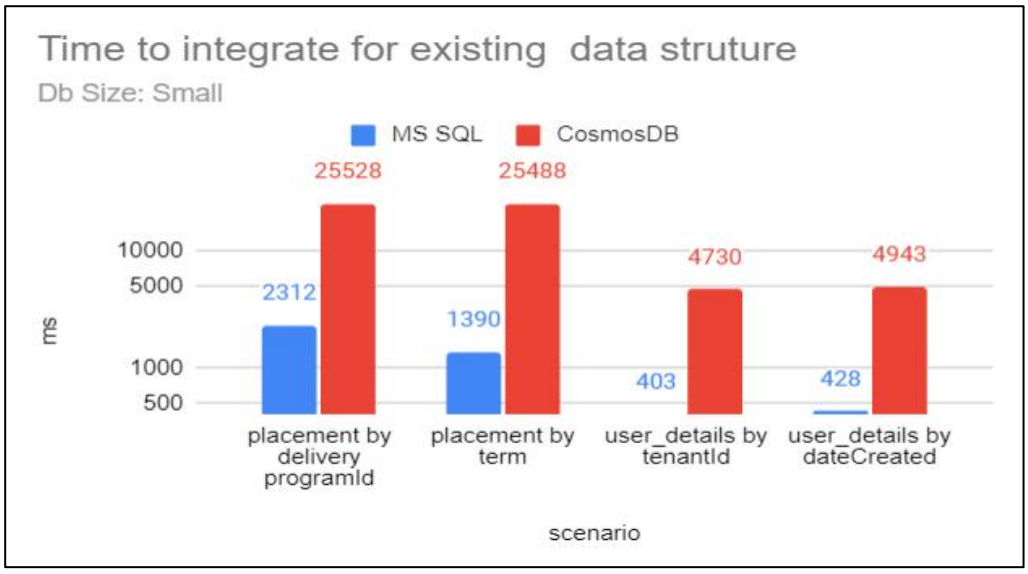

Рисунок 4.6 - Інтеграція в існуючу структуру для малого розміру даних

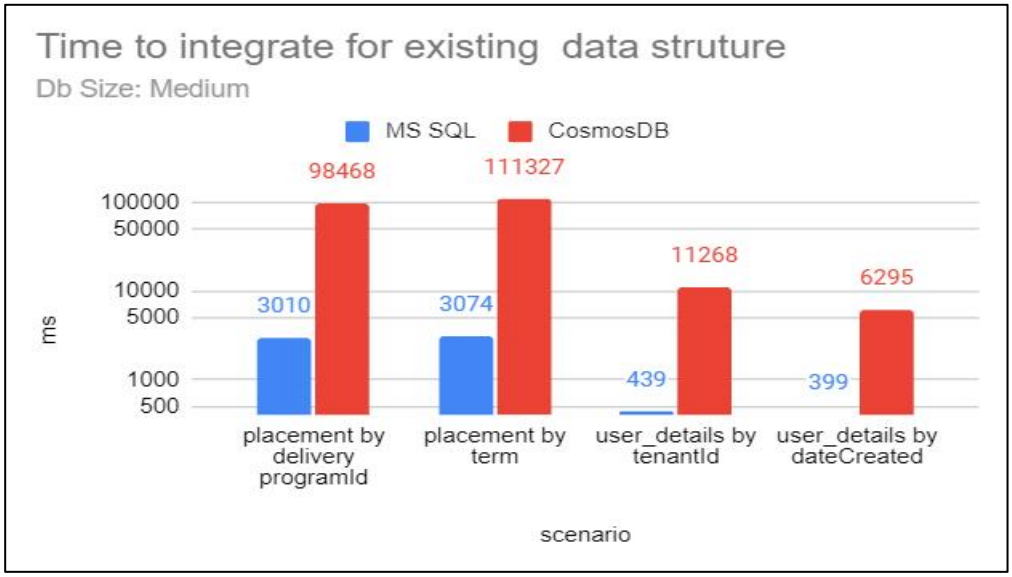

Рисунок 4.7 - Інтеграція в існуючу структуру для середнього розміру даних

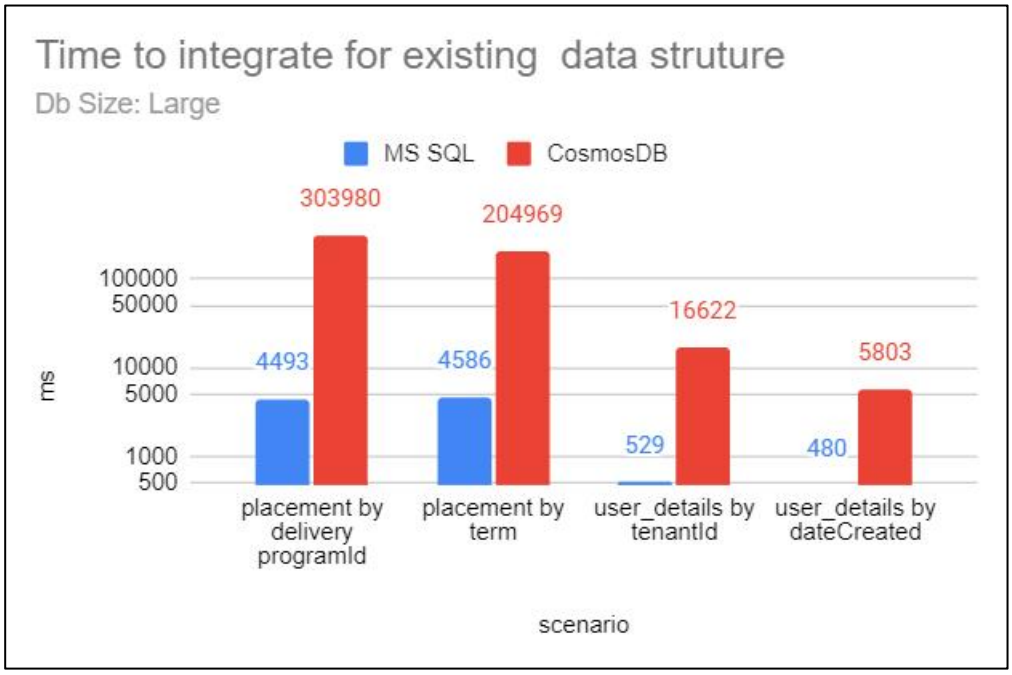

Рисунок 4.8 - Інтеграція в існуючу структуру для великого розміру даних

Проведене дослідження з точки зору інтеграції в існуючу структуру даних показало значну перевагу MS SQL Server. В цілому, якщо розмір даних таблиць малий, то більш пріоритетним використовувати Cosmos DB з його простотою виконання. Але ж якщо розмір даних перевищує 300Мб або 200 000 записів у таблиці, то міграція з використанням методів порціювання MS SQL Server однозначно краще.

4.1.4 Дослідження відносно метрики часу обчислювальних затрат та очікування

Для кожного зі сценаріїв порціювання було виконано раніше розроблені запити. Для сценаріїв з розподілом даних місця субсидії — select1, count1, update1, insert1. Для сценаріїв з розподілом даних користувача — select2, count2, update2, insert2, та замір часу виконання та очікування, для MS SQL та CosmosDB. Відповідні запити та сценарії були відпрацьовані на таблицях з великим (1200 Мб), середнім (600 Мб) та малим (100 Мб) розміром (див. табл. 4.4 - 4.6).

Таблиця 4.4 - Замір часу для малого розміру даних (100 Мб)

Сценарій	Тип команди	MS SQL		CosmosDB	
		Execution time, ms	Wait time, ms	Execution time, ms	Wait time, ms
Дані користувача за підпрограмою	select2	739	109	242	60
	count2	7	4	224	29
	update2	20	3	221	13
	insert2	7	3	221	30
Дані користувача за днем створення	select2	12	2	220	6
	count2	1	3	301	5
	update2	6	10	225	14
	insert2	4	20	221	7
Місце субсидії за програмою фінансування	select1	333	139	312	1
	count1	9	10	309	5
	update1	36	11	225	14
	insert1	4	4	312	6
Місце субсидії за роком фінансування	select1	333	136	322	4
	count1	10	3	316	6
	update1	40	3	226	14
	insert1	5	6	221	3

Для всіх сценаріїв запит get є найповільнішим. Запити на запис даних потребує меншу часу. CosmosDb в рамках даного дослідження в більшості випадків працює повільніше.

Таблиця 4.5 - Середній розмір (600 Мб). Замір часу

Сценарій	Тип команди	MS SQL		CosmosDB	
		Execution time, ms	Wait time, ms	Execution time, ms	Wait time, ms
Дані користувача за підпрограмою	select2	16	1	242	6
	count2	2	6	224	20
	update2	10	20	221	11
	insert2	4	30	221	11
Дані користувача за днем створення	select2	14	20	220	1
	count2	2	11	30	40
	update2	11	32	225	12
	insert2	3	45	221	13

Продовження таблиці

Сценарій	Тип команди	MS SQL		CosmosDB	
		Execution time, ms	Wait time, ms	Execution time, ms	Wait time, ms
Місце субсидії за програмою фінансування	select1	941	577	315	11
	count1	13	23	301	10
	update1	66	32	225	23
	insert1	4	50	221	73
Місце субсидії за роком фінансування	select1	1042	669	345	53
	count1	16	62	350	22
	update1	70	100	230	23
	insert1	4	245	220	230

Таблиця 4.6 - Повний розмір (1200 Мб). Замір часу

Сценарій	Тип команди	MS SQL		CosmosDB	
		Execution time, ms	Wait time, ms	Execution time, ms	Wait time, ms
Дані користувача за підпрограмою	select2	106	1	323	89
	count2	10	10	323	299
	update2	130	100	333	136
	insert2	30	2	323	38
Дані користувача за днем створення	select2	210	12	321	52
	count2	40	53	324	501
	update2	180	32	324	14
	insert2	50	45	332	11
Місце субсидії за програмою фінансування	select1	2059	1258	3000	1503
	count1	270	324	233	263
	update1	960	245	124	245
	insert1	40	532	245	605
Місце субсидії за роком фінансування	select1	2084	1279	2435	5252
	count1	310	245	354	167
	update1	101	243	300	844
	insert1	40	235	600	546

Було проведено дослідження з точки зору часу обчислювальних затрат та очікування, що показало певну перевагу зі сторони MS SQL.

4.2 Оцінка якості та вироблені рекомендації стосовно порціювання

Вибір методу порціювання як методу масштабування систем є дуже специфічним та залежить від дуже багатьох факторів. Серед них є специфіка бізнес процесів, особливість інфраструктури програмної системи та технології, що використовуються в ній.

В рамках кваліфікаційної роботи було порівняно методи порціювання для SQL та NoSQL систем. Представниками є MS SQL Server та CosmosDB.

Методи порціювання для CosmosDB найкраще підійдуть для систем з малим розміром даних (до 200 Мб), та систем, які не мають в вимогах високу ефективність. База даних CosmosDB виявилась ефективною з точки зору простоти інтерфейсу та меншої кількості обмежень при застосуванні порціювання. Також важливо підкреслити ефективність рівномірного розподілу даних порціювання. Проте час інтегрування в існуючу структуру даних є досить великим, на порядок вище, ніж в MS SQL Server. Час виконання запитів не дав суттєвого приросту та в цілому уступає часу виконання запитів для MS SQL Server, це стосується і операцій запису даних.

Методи порціювання MS SQL Server добре проявили себе з точки зору часу виконання запитів та максимально швидкий час для інтегрування в існуючу систему. Тому потеційно краще використовувати для проектів з великим розміром даних та з чутливими вимогами щодо часу обробки запитів. Проте для узгодження існуючих залежностей та зміні в структурі необхідно проробити велику роботу та задіяти не 1 скрипт, тому є більша ймовірність допустити помилки при реалізації. Ще мінусом є те, що не можливо задіяти порціювання транзакцією, так як змінюється сама структура бази даних (не тільки таблиця). Також мінусом є те, що залучити метод порціювання можна лише з детермінованим числом сегментів, та при додаванні нового сегменту необхідно здійснювати додаткову маніпуляцію, чого не було виявлено в CosmosDB.

ВИСНОВКИ

В ході виконання кваліфікаційної роботи були проаналізована проблемна область порціювання даних для масштабування розподілених систем та досліджені особливості методів порціювання даних для СУБД MS SQL Server та CosmosDB.

Було опрацьовано наступні питання:

– проведено аналіз проблемної області дослідження та розроблена постановка задачі;

– сплановано експериментального дослідження (спроектовано схеми БД, розроблено запити);

– спроектовано середовища для експерименту;

– проведено експеримент та аналіз результатів.

Для кваліфікаційної роботи була обрана реальна прикладна область з обрахунками фінансових операцій та формуванням звітності в сфері процесів працевлаштування на базі бюджету держзамовлень, в контексті якої була розроблена логічна модель бази даних.

В ході дослідження було складено план експериментального дослідження, де були виявленні основні метрики якості, а саме рівномірність розподілення, відповідність операцій даних для сегментів, час інтеграції в існуючу структуру даних, час обчислювальних затрат, час очікування. Було реалізовані сценарії порціювання місця субсидії за роком фінансування та за програмою постачання, а також сценарій з розподілом даних користувача за внутрішнім ідентифікатором програми, та за вищим навчальним закладом для двох баз даних.

Було проведено замір кожної з метрик для 2 баз даних на 3 різних наборах даних (великий об’єм 1200 Мб, середній 600 Мб та малий 100 Мб). Було впроваджено скрипти для інтеграції в існуючу структуру даних.

Було отримано результати дослідження та сформована оцінка якості та рекомендації стосовно порціювання.

За результатами дослідження було опубліковано тези «Дослідження методів порціювання баз даних для масштабування програмних систем» на 27-й Міжнародний молодіжний форум «Радіоелектроніка та молодь в XXI ст.» [21] (див. додаток Г).

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Springer link. Data partitioning. URL: https://link.springer.com/referenceworkentry/10.1007/978-0-387-39940-9_688/ (дата звернення 01.12.2022).

2. Afanasieva, I., Golian, N., Hnatenko, O., Daniiel, Y., & Onyshchenko, K. (2019). Data exchange model in the internet of things concept. Telecommunications and Radio Engineering (English Translation of Elektrosvyaz and Radiotekhnika), 78(10), 869-878. doi:10.1615/TelecomRadEng.v78.i10.30.

3. Mazurova, O. NOSQL DATABASE LOGIC DESIGN METHODS FOR MONGODB AND NEO4J / Mazurova, O., Syvolovskyi, I., Syvolovska, O. Innovative technologies and scientific solutions for industries, (2 (20), pp. 52-63. Doi: 10.30837/ITSSI.2022.20.052.

4. TestPrep. Horizontal, Vertical and Functional partitioning. URL: https://www.testpreptraining.com/tutorial/horizontal-vertical-and-functional-data-partitioning/ (дата звернення 02.12.2022).

5. Minimum replication min-cut partitioning. (б. д.). Home Page. https://doi.org/10.1109/43.662685 (дата звернення 02.15.2022).

6. Microsoft. Partition overview. URL: https://learn.microsoft.com/en-us/azure/cosmos-db/partitioning-overview (дата звернення 02.12.2022).

7. Implement a non-relational data model - Training. (б. д.). Microsoft Learn: Build skills that open doors in your career. https://learn.microsoft.com/en-us/training/modules/implement-non-relational-data-model/ .

8. Partitioned tables and indexes - SQL Server, Azure SQL Database, Azure SQL Managed Instance. (б. д.). Microsoft Learn: Build skills that open doors in your career. https://learn.microsoft.com/en-us/sql/relational-databases/partitions/partitioned-tables-and-indexes?view=sql-server-ver16 .

9. Measuring the performance of database object horizontal fragmentation schemes. (б. д.). Home Page. https://doi.org/10.1109/IDEAS.1999.787292 .

10. Microsoft. Create Partition Function overview. URL: https://learn.microsoft.com/en-us/sql/t-sql/statements/create-partition-function-transact-sql?view=sql-server-ver16 (дата звернення 03.12.2022).

11. TOPDB Top Database index. URL: https://pypl.github.io/DB.html (дата звернення 03.12.2022).

12. MySQL Partitioning - w3resource. URL: https://www.w3resource.com/mysql/mysql-partition.php (дата звернення 03.12.2022).

13. Recommendations and Guidelines on configuring disk partitions for SQL Server - Microsoft Support. URL: https://support.microsoft.com/en-us/topic/recommendations-and-guidelines-on-configuring-disk-partitions-for-sql-server-a25faa94-509f-370b-0975-ee0b26541aa9 (дата звернення 03.12.2022).

14. PostgreSQL. Table Partitioning. URL: https://www.educba.com/postgresql-table-partitioning/ (дата звернення 04.12.2022).

15. Data Partitioning with Chunks — MongoDB Manual. URL: https://www.mongodb.com/docs/manual/core/sharding-data-partitioning/ (дата звернення 04.12.2022).

16. Azure Cosmos DB partitioning design patterns – Part 1 | Azure Blog and Updates | Microsoft Azure. URL: https://azure.microsoft.com/en-in/blog/azure-cosmos-db-partitioning-design-patterns-part-1/ (дата звернення 04.12.2022).

17. Request Units as a throughput and performance currency - Azure Cosmos DB. Microsoft Learn: Build skills that open doors in your career. https://learn.microsoft.com/en-us/azure/cosmos-db/request-units (дата звернення 04.12.2022).

18. Azure cosmos DB data migration tool | azure updates | microsoft azure. (б. д.). Cloud Computing Services | Microsoft Azure. https://azure.microsoft.com/updates/documentdb-data-migration-tool/ (дата звернення 08.01.2023).

19. Azure Cosmos DB modeling and partition data using real world example. URL: https://learn.microsoft.com/en-us/azure/cosmos-db/nosql/how-to-model-partition-example (дата звернення 08.01.2023).

20. How to automate table partitioning in MS SQL Server. Main strategies. URL: https://www.sqlshack.com/how-to-automate-table-partitioning-in-sql-server/ (дата звернення 16.01.2023).

21. Ковалевич Б. І. Дослідження методів порціювання даних для масштабування програмних систем // 27-й Міжнародний молодіжний форум «Радіоелектроніка та молодь у XXI столітті». Зб. матеріалів форуму. Т. 6. – Харків: ХНУРЕ. 2023., с. 343-344.