МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки та програмної інженерії
Кафедра інженерії програмного забезпечення

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
Катерина НЕСТЕРЕНКО
“____”_____2023 р.

КВАЛІФІКАЦІЙНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСНИКА ОСВІТНЬОГО СТУПЕНЯ
МАГІСТРА

Тема: “Теоретико-методичні засади розробки OLAP систем управління базами даних”

Виконавець: Михайлов Вячеслав Андрійович

Керівник: к.т.н доцент Шибицька Наталія Миколаївна

Нормоконтролер: асистент Кравченко Ольга Сергіївна

Київ 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки та програмної інженерії
Кафедра інженерії програмного забезпечення
Освітній ступінь магістр
Спеціальність 121 Інженерія програмного забезпечення
Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Катерина НЕСТЕРЕНКО
"___" _____ 2023 р

ЗАВДАННЯ
на виконання кваліфікаційної роботи студента
Михайлова Вячеслава Андрійовича

1. Тема кваліфікаційної роботи: «Теоретико-методичні засади розробки OLAP систем управління базами даних»
затверджена наказом ректора від 29.09.2023 р. № 1994/ст

2. Термін виконання проекту: з 02.10.2023 р. до 31.12.2023 р.

3. Вихідні дані до роботи: програмний продукт розробити за допомогою .NET Core.

4. Зміст пояснювальної записки:
   1. Дослідження предметної області. Первісний аналіз.
   2. Дослідження вимог до системи.
   3. Дослідження алгоритмів. Методичні вказівки з вібору та реалізації алгоритмів.
   4. Проектування архітектури. Прототип системи.

5. Перелік обов'язкових слайдів презентації:
   1. Визначення OLAP баз даних
   2. Функціональні можливості OLAP баз даних
   3. Методи індексації в OLAP базах даних
   4. Методи зберігання, реплікації та секціонування в OLAP базах даних
   5. Проектування OLAP бази даних
   6. Демонстрація роботи модулів програми

6. Календарний план-графік

№ пор	Завдання	Термін виконання	Відмітка про виконання
1.	Розробка та затвердження графіка роботи.	02.10.2023 – 07.10.2023	
2.	Підготовка та написання 1 розділу. Відсилка керівнику	07.10.2023 – 31.10.2023	
3.	Підготовка та написання 2 розділу. Відсилка керівнику	01.11.2023 – 14.11.2023	
4.	Написання 3 розділу, представлення керівнику	15.11.2023 – 25.11.2023	
5.	Написання 4 розділу, представлення керівнику	26.11.2023 – 05.12.2023	
6.	Загальне редагування та друк пояснювальної записки, графічного матеріалу	06.12.2023	
7.	Проходження нормо-контролю, перепліт пояснювальної записки.	07.12.2023	
8.	Розробка тексту доповіді. Оформлення графічного матеріалу для презентації. Отримання відгуку керівника.	07.12.2023 – 10.12.2023	
9.	Передзахист (наявність віддрукованої ПЗ, презентації, позитивного відгуку керівника). Отримання рецензії.	11.12.2023 – 17.12.2023	
9.	Підготовка матеріалів для передачі секретарю ДЕК (ПЗ, ГМ, CD-R з електронними копіями ПЗ, ГМ, презентації, відгук керівника, рецензія, довідка про успішність, папка: уточнити у секретаря ДЕК)	18.12.2023 – 24.12.2023	
10	Захист кваліфікаційної роботи	25.12.2023 – 31.12.2023	

Дата видачі завдання 02.10.2023

Керівник: к.т.н доцент Наталія ШИБИЦЬКА

Завдання прийняв до виконання: Вячеслав МИХАЙЛОВ

РЕФЕРАТ

Пояснювальна записка до дипломної роботи «Теоретико-методичні засади розробки OLAP систем управління базами даних»: 97 сторінок, 24 рисунки, 5 таблиць, 10 інформаційних джерел.

БАЗИ ДАНИХ, СИСТЕМИ УПРАВЛІННЯ БАЗАМИ ДАНИХ, РОЗПОДІЛЕНІ БАЗИ ДАНИХ, ІНДЕКСАЦІЯ ДАНИХ, OLAP БАЗИ ДАНИХ.

Об’єкт дослідження – OLAP системи управління базами даних.

Мета роботи – формування теоретико-методичних засад розробки OLAP систем управління базами даних.

Метод дослідження – порівняння об’єкту дослідження із традиційними транзакційними базами даних, визначення та практичне тестування основних алгоритмів, на яких будуються обидва типи баз даних.

В процесі дослідження було проведено аналіз предметної області, знайдено фактори, що сприяли появі OLAP баз даних. На основі цих факторів було сформовано вимоги та ТЗ до об'єкта дослідження, після чого було проведено глибокий порівняльний аналіз методів реалізації баз даних, наведено аргументовані рекомендації до вибору алгоритмів при реалізації OLAP баз даних, а також створено прототип такої системи.

Результати роботи можуть слугувати основою для вдосконалення існуючих реалізацій OLAP баз даних. Одним із ключових аспектів є вдосконалення алгоритмів, що використовуються для зберігання та індексації даних. Результати дослідження можуть бути безпосередньо впроваджені в деякі OLAP системи для підвищення їх продуктивності, швидкості відгуку на запити, а також для забезпечення більш ефективного використання ресурсів сервера.

ABSTRACT

Explanatory note for the thesis "Theoretical and methodological basics of developing OLAP database management systems": 97 pages, 24 figures, 5 tables, 10 references.

DATABASES, DATABASE MANAGEMENT SYSTEMS, DISTRIBUTED DATABASES, DATA INDEXING, OLAP DATABASES.

The object of research is OLAP database management systems.

The purpose of the work is to form theoretical and methodical basics of developing OLAP database management systems.

Research method involves comparing the research object with traditional transactional databases, defining and practically testing the main algorithms underlying both types of databases.

The research involved analyzing the subject area, identifying factors contributing to the emergence of OLAP databases. Based on these factors, requirements and specifications for the research object were formulated. A deep comparative analysis of database implementation methods was conducted. Well-founded recommendations were provided for choosing algorithms in OLAP database implementation, and a prototype of such a system was created.

The results could serve as a basis for improving existing implementations of OLAP databases. An essential aspect is enhancing the algorithms used for data storage and indexing. The research outcomes can be directly implemented in some OLAP systems to enhance their productivity, query response speed, and ensure more efficient server resource utilizatio

ЗМІСТ

ПЕРЕЛІК ПРИЙНЯТИХ СКОРОЧЕНЬ

БД – база даних;

СУБД – система управління базами даних;

OLAP - (англ. Online Analytical Processing – аналітична обробка у реальному часі) – тип обробки даних, який полягає у багатовимірному аналізі великих обсягів даних на високій швидкості у реальному часі;

OLTP – (англ. Online Transaction Processing – обробка транзакцій у реальному часі) –тип обробки даних, який полягає у виконанні кількох транзакцій, що відбуваються одночасно, наприклад, онлайн-банкінг, покупки, введення замовлень або надсилання текстових повідомлень;

SQL – (англ. Structured Data Language) - мова програмування для зберігання та обробки інформації в базі даних;

NoSQL – (англ. Not Only SQL - не тільки SQL) - це підхід до проектування бази даних, який дозволяє зберігати та надсилати запити до даних поза традиційними структурами реляційних баз даних;

ООП – об’єктно-орієнтоване програмування;

КОП – компонентно-орієнтоване програмування;

ТЗ – технічне завдання;

ПЗ – програмне забезпечення;

SIMD - (англ. Single-instruction-multi-data - одиночний потік команд, множинний потік даних) — це елемент класифікації згідно з таксономією Флінна для паралельних процесорів, де до багатьох елементів даних виконується одна або однакові команди.

ВСТУП

У сучасному інформаційному суспільстві здатність оперативно обробляти, аналізувати та інтерпретувати великі обсяги інформації стає критичною для успішності бізнесу, наукових досліджень, а також для ефективного управління різними сферами діяльності. Системи, які забезпечують гнучкий та динамічний аналіз даних, набувають особливої важливості.

Об'єктом дослідження цієї кваліфікаційної роботи є системи, що відповідають таким вимогам - OLAP (Online Analytical Processing) системи, які дозволяють проводити багатовимірний аналіз даних.

OLAP-системи є ключовим компонентом більшості сучасних корпоративних інформаційних систем. Вони надають можливість користувачам виконувати складні запити до баз даних у режимі реального часу, а також забезпечують інтерактивний доступ до інформації, що зберігається у великих базах даних. Втім, розробка та впровадження OLAP-систем вимагає глибокого розуміння як теоретичних, так і практичних аспектів, що стосуються їх структури, архітектури та принципів функціонування.

Тож актуальність дослідження теоретичних та методологічних основ розробки OLAP систем управління базами даних обумовлена декількома ключовими чинниками. По-перше, постійний ріст об'ємів даних, які зберігаються в електронному вигляді, вимагає від спеціалістів нових підходів до їх аналізу та обробки. По-друге, еволюція технологій баз даних веде до появи нових методологій та підходів до розробки систем, які повинні бути гнучкими, масштабованими та відповідати сучасним вимогам. По-третє, управління інформаційними ресурсами стає ключовою конкурентною перевагою для організацій, що підкреслює важливість якісного аналізу даних. Таким чином, дослідження теоретичних та методологічних аспектів розробки OLAP систем є не лише актуальним, але й надзвичайно важливим для розвитку сучасних інформаційних систем.

Враховуючи вищезазначене, цей проект намагається визначити ключові принципи, методи та технології, які можуть підвищити ефективність OLAP систем в контексті сучасних вимог до обробки та аналізу великих обсягів даних. Основні завдання, які стоять перед дослідженням, включають:

1. Аналіз існуючих теоретичних підходів до розробки OLAP систем.
2. Оцінка актуальних методологій та інструментів, які застосовуються в OLAP.
3. Ідентифікація маловивчених або недосліджених аспектів, що потребують додаткового вивчення.
4. Розробка та тестування прототипів на базі існуючих або удосконалених алгоритмів для підвищення ефективності OLAP систем.
5. Узагальнення результатів дослідження для формулювання рекомендацій та методологічних принципів для подальшої роботи в даній області.

В рамках проекту буде використано комплексну методику дослідження, яка комбінує як теоретичні, так і практичні методи:

1. Концептуалізація: Розробка або адаптація концептуальних моделей для систематизації знань з теми, визначення ключових концептів, термінів та зв'язків між ними.
2. Аналіз предметної області: Глибокий аналіз специфіки предметних областей, в яких OLAP системи найбільш активно використовуються (наприклад, бізнес-аналітика, наукові дослідження, управління проектами), для виявлення специфічних вимог та проблем.
3. Дослідження алгоритмів: Вивчення та аналіз алгоритмічної бази OLAP систем — методи компресії даних, алгоритми швидкого пошуку, методи оптимізації запитів тощо. Це також може включати розробку нових алгоритмів або адаптацію існуючих для підвищення ефективності OLAP систем.
4. Прототипізація: Розробка OLAP систем з метою перевірки теоретичних концептів, методологій та алгоритмів на практиці. Прототипи

допоможуть ідентифікувати потенційні слабкі місця, оцінити продуктивність, а також перевірити практичну застосовність розроблених методів і технік. Цей метод є особливо цінним для валидації наукових гіпотез і для забезпечення їх практичної релевантності.

5. Експериментальний метод: Проведення експериментів на прототипах OLAP систем з метою перевірки теоретичних висновків та моделей на практиці.

Застосування цих методів дослідження дозволить не тільки глибше зрозуміти існуючі проблеми і виклики в розробці OLAP систем, але і сформувати науково-обгрунтовану базу для подальших інновацій у цій області.

Окрім теоретичного дослідження, у цьому дипломному проекті описано процес проектування та створення рпототипу OLAP системи із використанням алгоритмів та методів, обраних завдяки попередньому аналізу. Для розробки прототипу використано платформу .NET Core.

Практична значимість даного проекту може мати широкий спектр застосувань в різних сферах діяльності — від бізнесу до наукових досліджень. Основні результати праці можуть слугувати базою для вдосконалення існуючих моделей OLAP баз даних. Одним з ключових аспектів є вдосконалення алгоритмів, що використовуються для зберігання та індексації даних. Результати дослідження можуть бути безпосередньо впроваджені в деякі OLAP системи для підвищення їх продуктивності, швидкості відгуку на запити, а також для забезпечення більш ефективного використання ресурсів сервера.

Крім того, дана робота може стати важливим методологічним посібником для розробників баз даних, системних аналітиків, науковців та спеціалістів в області Big Data. Результати дослідження мають потенціал для використання в навчальних курсах по базам даних та OLAP технологіях, а також в спеціалізованих тренінгах для IT-фахівців. У подальшому, ці результати можуть стати основою для розробки нових OLAP продуктів або модулів, що інтегруються в існуючі інформаційні системи, забезпечуючи тим самим стратегічну конкурентну перевагу для

організацій, які їх використовують. Отже, практична значимість даної наукової праці є очевидною і багатогранною.

РОЗДІЛ 1
ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ. ПЕРВІСНИЙ АНАЛІЗ

Основною метою цього етапу є надати чітке і всеосяжне розуміння предметної області OLAP систем управління базами даних. Розділ прагне охарактеризувати існуючий стан знань та практик в даній області, ідентифікувати ключові проблеми, маловивчені аспекти та можливі напрямки для подальших досліджень. Це забезпечить фундаментальну базу для подальшого аналізу, розробки та вдосконалення OLAP систем. Даний етап повинен забезпечити узгодженість обох сторін (замовника та розробника), щодо найбільш важливих етапів розробки: стадії аналізу, проектування, реалізації, документування, впровадження й промислової експлуатації.

Основні результати етапу дослідження предметної області:

- Проведено аналіз історії розвитку OLAP технологій, їх впливу на сфери бізнес-аналітики та управління великими даними.
- Описані основні теоретичні концепції, які лежать в основі OLAP.
- Створено техніко-економічну характеристику предметної області.
- Ідентифіковані маловивчені або недосліджені аспекти в OLAP, які можуть бути предметом подальших наукових досліджень.
- Оцінено застосування OLAP в різних сферах діяльності, з вказівкою на специфічні виклики та перспективи в кожному випадку.

Метою цього розділу є можливість можна зробити висновок чи є OLAP системи важливою частиною сучасної екосистеми аналітики даних, та чи дозволяють вони ефективно управляти великими обсягами інформації, видобувати з неї цінну інформацію і тим самим підвищувати ефективність бізнес-процесів.

1.1. Динаміка розвитку OLAP технологій

Одним із важливих напрямків у сфері інформаційних технологій є аналітичні бази даних, де OLAP (Online Analytical Processing) відіграє ключову роль. З перших

днів свого виникнення в кінці 1980-х і до сьогодні, OLAP системи пережили декілька етапів розвитку, від простих до складних та динамічних рішень.

1.1.1. Розвиток OLAP технологій: від простих систем до сучасних рішень

OLAP технології вперше з'явилися як засоби для бізнес-аналітики, дозволяючи аналізувати невеликі набори даних в простих багатовимірних структурах. Відомим прикладом раннього OLAP інструменту є Lotus Improv, який вперше був представлений у 1991 році. Прості структури і нескладні алгоритми дозволяли використовувати OLAP в основному для фінансового моделювання та звітності.

На початку 2000-х, з ростом об'ємів даних та розвитком архітектур баз даних, OLAP технології почали еволюціонувати. З'явилися поняття, такі як "широкий OLAP" (WOLAP), "реляційний OLAP" (ROLAP) та "гібридний OLAP" (HOLAP), що забезпечили нові можливості для обробки та аналізу більших об'ємів даних. Відбувалося активне впровадження OLAP в різних сферах: від фінансів до наукових досліджень.

Останнім часом, основними напрямками розвитку OLAP є адаптація до обробки великих даних, інтеграція з машинним навчанням та штучним інтелектом, а також аналіз у реальному часі. Сучасні OLAP системи можуть працювати з розподіленими базами даних, обробляти запити в реальному часі і навіть здійснювати прогнозування.

1.1.2. Сучасні реалізації комбінації OLAP систем та баз даних

В останні роки, OLAP технології зазнали значущих змін, особливо з появою хмарних рішень та розподілених систем зберігання даних. Сьогодні OLAP не лише інструмент для бізнес-аналітики, але і ключова складова в сучасних архітектурах баз даних. З розвитком хмарних технологій, багато компаній почали переносити свої OLAP системи в хмару. Платформи, такі як Google BigQuery OLAP, Amazon Redshift та Microsoft Azure Analysis Services, надають можливість швидко масштабувати ресурси та забезпечити високу доступність.

Сучасні OLAP системи часто використовують колоночні бази даних для оптимізації швидкості запитів. Системи, такі як ClickHouse або Apache Druid, забезпечують високу продуктивність при роботі з великими наборами даних, а з ростом об'ємів даних та потреб в реальному часі аналізу, OLAP системи все більше адаптуються до розподілених архітектур. Такі системи можуть працювати на кластерах з декількох серверів, що дозволяє паралельно обробляти запити та забезпечує високу швидкість відгуку.

Останнім трендом в OLAP технологіях є інтеграція з NoSQL базами даних. Такі системи, як MongoDB або Cassandra, можуть бути використані для зберігання неструктурованих або напівструктурованих даних, які потім можуть бути аналізовані за допомогою OLAP інструментів.

Враховуючи все вищевказане, сучасні OLAP бази даних представляють собою динамічні, гнучкі та масштабовані рішення, які можуть адаптуватися до широкого спектра задач та вимог.

1.1.3. Вплив розвитку OLAP-технологій на сфери бізнес-аналітики та управління великими даними

OLAP системи зробили свій внесок у сферах аналізу даних, та в цілому на сферу бізнесу, а саме:

- Однією з найбільших переваг OLAP технологій є можливість швидкого доступу до інформації та аналізу даних. Це зробило процеси прийняття рішень в бізнесі значно ефективнішими, забезпечуючи оперативність та вчасність дій.
- Історично OLAP системи стали більш дружніми для користувачів, що не є IT-спеціалістами. Це призвело до демократизації даних, коли менеджери різних рівнів можуть самостійно виконувати аналіз без завдань для IT-відділу.
- Розвиток OLAP технологій дозволив здійснювати більш складні види аналізу, включаючи трендовий, прогнозний, та мультиваріативний аналізи. Це значно розширило можливості бізнес-аналітики.

- Сучасні OLAP системи, особливо розподілені та хмарні, можуть ефективно обробляти терабайти та петабайти даних. Це особливо важливо в епоху Big Data, коли об'єми даних продовжують зростати.
- Колоночні бази даних і спеціалізовані алгоритми індексації дозволили оптимізувати зберігання даних, зменшуючи вартість та підвищуючи ефективність.
- OLAP системи все частіше інтегруються з NoSQL базами даних та потоковими платформами (наприклад, Kafka), що дозволяє об'єднувати структуровані та неструктуровані дані для комплексного аналізу.

В цілому, історія розвитку OLAP технологій суттєво вплинула на методи управління великими об'ємами даних та на практику бізнес-аналітики. Їх вплив можна вважати перетворювальним, оскільки вони змінили не лише технічну сторону обробки даних, але і організаційну культуру, зробивши дані доступними та корисними на всіх рівнях бізнесу.

1.2. Основні теоретичні концепції

Цей дипломний проект має на меті глибоке вивчення основних теоретичних концепцій, які лежать в основі OLAP технологій. Буде розглянуто ключові принципи многовимірного аналізу, ієрархій, агрегації даних, а також різні види запитів та операцій, які можна виконати за допомогою OLAP систем. Це забезпечить чітке розуміння потенціалу та обмежень цих систем, а також їх можливостей для різних застосувань.

Розуміння теоретичних концепцій OLAP не лише сприяє ефективному використанню цих систем, але і є важливим для їх подальшого розвитку та вдосконалення. Вивчення цих концепцій дозволить нам оцінити потенціал OLAP для решення сучасних проблем обробки та аналізу великих обсягів даних, а також їх можливі шляхи еволюції.

1.2.1. Концепція багатовимірного аналізу

Багатовимірний аналіз є однією з ключових характеристик OLAP (Online Analytical Processing) систем, що відзначає їх від традиційних реляційних баз даних. Центральним елементом багатовимірного аналізу є концепція "куба даних" (data cube), яка репрезентує набір даних у многовимірному просторі. Кожна вісь цього куба представляє окремий "вимір" аналізу — наприклад, час, географічні регіони, продукти чи категорії користувачів.

Ця многовимірна структура надає велику гнучкість при виконанні аналітичних запитів. Користувач може "зрізати" куб по будь-яким двом або більше вимірам, що дозволяє отримувати агреговані дані за різними критеріями. Наприклад, можна легко вивчити продажі конкретного продукту в різних регіонах протягом певного періоду часу.

Ще однією перевагою багатовимірного аналізу є можливість деталізації і агрегації даних завдяки ієрархічним структурам. Кожен вимір може містити кілька рівнів ієрархії. Так, вимір "Час" може включати рівні "День", "Місяць", "Квартал", і "Рік". Це дозволяє користувачам легко проводити аналіз на різних рівнях гранулярності, використовуючи операції "drill-down" (розкручування до деталей) та "roll-up" (агрегація на вищий рівень).

Багатовимірний аналіз є фундаментальним для OLAP систем, оскільки він надає широкі можливості для вивчення та інтерпретації даних, що є критично важливим в сферах, де потрібно швидко приймати обґрунтовані рішення. Він покращує інтуїтивність і зрозумілість аналізу, забезпечуючи при цьому гнучкість і високу продуктивність обробки запитів.

1.2.2. Концепція ієрархії та агрегації в OLAP системах

Однією з основних характеристик систем OLAP (Online Analytical Processing) є концепція ієрархії, яка створює структурований підхід до аналізу даних. Ієрархії дозволяють розглядати дані на різних рівнях деталізації в рамках одного виміру. Наприклад, вимір "Час" може мати ієрархічну структуру, що складається з рівнів

"День", "Місяць", "Квартал" і "Рік". Це надає аналітикам можливість легко переходити від загального огляду даних до їх деталізованого вивчення.

Агрегація є тісно пов'язаною з ієрархією і є ключовим аспектом OLAP систем. Цей процес включає у себе комбінування даних на нижчому рівні ієрархії для отримання підсумкових значень на вищому рівні. Агрегація може бути виконана за допомогою різних математичних операцій, таких як сума, середнє, максимум, мінімум тощо.

Використання агрегації дозволяє користувачам OLAP систем швидко отримувати зведену інформацію, що є критично важливим для стратегічного планування та прийняття рішень. Наприклад, менеджер може швидко побачити загальні продажі по всіх магазинах, а потім "пробуритися" до даних конкретного регіону або навіть окремого магазину.

Ієрархія та агрегація взаємно доповнюють одна одну в OLAP системах. Ієрархічна структура надає контекст для агрегованих даних, тоді як агрегація дозволяє ефективно взаємодіяти з цією структурою. За допомогою операцій "drill-down" (зріз вниз по ієрархії) та "roll-up" (зріз вгору), користувачі можуть гнучко переходити між різними рівнями ієрархії, отримуючи при цьому агреговану інформацію, що є найбільш релевантною для конкретного аналізу.

В цілому, концепції ієрархії та агрегації є фундаментальними для функціональності OLAP систем. Вони не тільки спрощують процес аналітичного аналізу, але і роблять його більш інтуїтивно зрозумілим

1.2.3. Концепція материалізації підрахунків

Матеріалізація підрахунків є однією з ключових концепцій в системах OLAP (Online Analytical Processing), спрямованих на оптимізацію процесу аналітичного запитання. В epi Big Data, коли об'єми інформації постійно ростуть, швидкість доступу до агрегованих даних стає критичною. Матеріалізація підрахунків полягає в попередньому розрахунку та збереженні агрегатів або інших статистичних мір в базі даних, що дозволяє швидко відповідати на запитання користувачів.

Суть матеріалізації полягає в тому, що вирази, які часто використовуються в запитах, підраховуються заздалегідь і зберігаються в спеціалізованих структурах даних, таких як матеріалізовані види, куби данних або інші кеш-структури. Коли аналітик або інший кінцевий користувач виконує запит, система спершу перевіряє, чи є підраховані дані уже доступні. Якщо так, результати можна швидко отримати, майже миттєво.

На відміну від звичайної агрегації на льоту, матеріалізація підрахунків вимагає попередньої обробки і збереження результатів в базі даних. Це може вимагати додаткових ресурсів зберігання і потужностей обчислень для попередньої обробки, але в залежності від сценарію, вигода в швидкості може переважити ці недоліки.

Однією з проблем матеріалізації є необхідність утримання актуальності агрегованих даних. Якщо базові дані змінюються, матеріалізовані підрахунки також повинні бути оновлені, що може бути викликом з точки зору процесу управління даними.

Матеріалізація підрахунків є могутньою концепцією в OLAP системах, що може значно підвищити швидкість аналітичних запитань. Це особливо важливо в сучасних умовах, коли аналітика стає все більш і більш централізованою в стратегічному плануванні та прийнятті рішень. Зважаючи на виклики та обмеження, ефективне використання матеріалізації вимагає глибокого розуміння не тільки самої концепції, але і специфіки даних та запитів, які будуть виконуватися.

1.2.4. Концепція індексації та кешування

Кешування та індексація є двома ключовими механізмами, які допомагають оптимізувати роботу OLAP (Online Analytical Processing) систем. Вони дозволяють забезпечити швидкий доступ до даних для виконання складних аналітичних запитань, що є критичним для бізнес-аналітики, наукових досліджень та інших сценаріїв, де потрібна швидка і точна обробка великих наборів даних.

Кешування в OLAP системах — це техніка, при якій результати запитів тимчасово зберігаються в оперативній пам'яті для подальшого використання. Це

дозволяє зменшити навантаження на систему при повторних запитах та забезпечити швидкий доступ до часто запитуваних даних. Кешування часто використовується в комбінації з іншими методами оптимізації, такими як матеріалізація підрахунків та індексація.

Індексація є іншим критично важливим аспектом проектування OLAP систем. Індекси допомагають швидко знаходити потрібні рядки або блоки даних в базі, значно зменшуючи обсяг даних, які потрібно сканувати. Індекси можуть бути створені на основі різних атрибутів та можуть мати різні структури, такі як B-дерева, хеш-таблиці, bitmap індекси та ін.

Обидва механізми часто використовуються разом для досягнення максимальної ефективності. Наприклад, індекс може бути використаний для швидкого знаходження даних, які потім кешуються для подальших запитів. Або кешовані агрегати можуть бути індексовані для забезпечення ще швидшого доступу.

Однак обидва підходи мають свої виклики. Кешування вимагає ефективного механізму інвалідації для підтримки актуальності даних. Індексація може призвести до збільшення обсягу зберігання та складність управління базою даних. Крім того, індекси повинні бути оновлені при зміні базових даних.

Кешування та індексація є фундаментальними техніками в OLAP системах, які вирішують проблему швидкісного доступу до даних. Хоча вони не є без своїх викликів та обмежень, їхнє ефективне використання може значно підвищити продуктивність системи, покращуючи користувацький досвід і забезпечуючи можливість проведення більш складних аналізів в короткі часові рамки.

1.2.5. Концепція роздільного аналізу

Один з найбільш характерних і розпізнаваних механізмів аналізу в OLAP (Online Analytical Processing) системах — це так званий "роздільний аналіз" (Slice-and-Dice). Ця концепція дозволяє користувачам "розрізати" багатовимірний куб даних по одній або декількох вісях, щоб зосередити увагу на конкретній підмножині даних, а також "роздробити" її для детального аналізу.

Уявимо, що у нас є багатовимірний куб даних, який включає виміри, такі як час, географічна локація, і продукт. За допомогою роздільного аналізу, користувач може вибрати певний "шматок" даних для аналізу, наприклад, продажі в Києві за липень 2023 року. Потім цей "шматок" можна "роздробити" на ще менші частини, аналізуючи продажі по окремих категоріях продуктів або навіть конкретних SKU.

Такий підхід корисний в багатьох сферах: від роздрібної торгівлі до здоров'я, від фінансів до логістики. Наприклад, в роздрібній торгівлі аналітики можуть використовувати роздільний аналіз для виявлення сезонних трендів або для аналізу ефективності маркетингових кампаній. В області здоров'я це може бути використано для аналізу розподілу захворювань за регіонами, віковими групами або іншими критеріями.

З технічного погляду, роздільний аналіз часто реалізований через мови запитів, такі як MDX (Multidimensional Expressions) або схожі. Це дозволяє забезпечити високу гнучкість і можливість персоналізації аналізу.

Концепція роздільного аналізу є однією з ключових можливостей OLAP систем, яка забезпечує глибокий аналітичний інсайт через інтерактивну роботу з багатовимірними даними. Її значущість не можна переоцінити, особливо в сучасному світі, де швидкий доступ до точної інформації може бути вирішальним фактором успіху в бізнесі та інших сферах.

1.3. Аналіз існуючих розробок

В умовах експоненціального росту обсягів даних і збільшення їх складності, OLAP (Online Analytical Processing) системи відіграють ключову роль в області бізнес-аналітики, наукових досліджень, управління ресурсами та інших сферах, де потрібна швидка та глибока аналіза даних. З цієї перспективи, аналіз існуючих розробок не просто важливий, але і необхідний для розуміння сильних та слабких сторін поточних OLAP систем, ідентифікації можливих напрямків для подальшого вдосконалення та інновацій.

Перш за все, аналіз допомагає оцінити ступінь зрілості технології, її доступність для різних типів користувачів та здатність задовольнити сучасні

потреби з обробки та аналізу даних. Це дозволяє вирішити, чи можна вважати поточні рішення ефективними, або ж чи існують прогалини, які вимагають подальшого дослідження та розробки.

Другою важливою причиною є необхідність уникнення "перевинаходу колеса". Зрозуміло, що розробка нових OLAP систем — це ресурсомісткий процес, який вимагає значних вкладень часу, зусиль та фінансів. Вивчення існуючих розробок дозволяє виявити вже вирішені проблеми, зосередитися на нових викликах та спрямувати ресурси на найбільш актуальні задачі.

Третє, аналіз існуючих розробок може виявити нові можливості для крос-дисциплінарних досліджень та партнерства між науковцями, розробниками та підприємцями. Особливо це актуально в контексті швидкого розвитку смежних технологій, таких як штучний інтелект, машинне навчання та Big Data.

З урахуванням згаданих факторів, аналіз існуючих розробок OLAP систем набуває особливого значення для формування комплексного та збалансованого погляду на стан та перспективи розвитку цієї важливої технології.

1.3.1. ClickHouse

ClickHouse є відкритим програмним забезпеченням для управління колоночними базами даних, яке спеціалізується на здійсненні швидких аналітичних запитів на великих обсягах даних. Воно активно використовується у ряді інтернет-сервісів для аналізу користувацької активності, моніторингу, обробки логів, і так далі.

Основна перевага ClickHouse полягає у його високій продуктивності і здатності ефективно виконувати запити на агрегацію даних. Система використовує колоночне зберігання даних, що дозволяє значно зменшити обсяг дискового простору, необхідного для зберігання даних, і збільшити швидкість зчитування.

ClickHouse добре масштабується горизонтально, що робить його привабливим рішенням для обробки великих наборів даних. Він також підтримує ряд розширених можливостей, таких як матеріалізовані погляди, віндові функції, та інші, які подальше покращують ефективність аналітичних запитів.

Система ClickHouse має велику активну спільноту і широкий спектр вбудованих інструментів для інтеграції з іншими системами управління базами даних та інструментами для візуалізації даних. Це робить його гнучким рішенням, яке може бути адаптовано для різних аналітичних потреб та бізнес-вимог.

Таким чином, ClickHouse являє собою потужний інструмент для аналітичних завдань, який може ефективно займатися обробкою великих обсягів даних в реальному часі, пропонуючи високу продуктивність і гнучкість для різних типів аналітичних завдань.

Особливості:

- Колонкова пам’ять для високого стиснення та швидкої роботи запитів.
- Розподілена та масштабована архітектура.
- Підтримка запитів SQL, включаючи складні аналітичні запити.
- Можливості обробки даних у реальному часі.

Переваги:

- Виняткова продуктивність запитів для аналітичних навантажень.
- Ефективна робота з великими обсягами даних.
- Підходить для аналітики та звітності в реальному часі.

Розглянемо діаграму мутації даних у ClickHouse на Рис. 1.1.

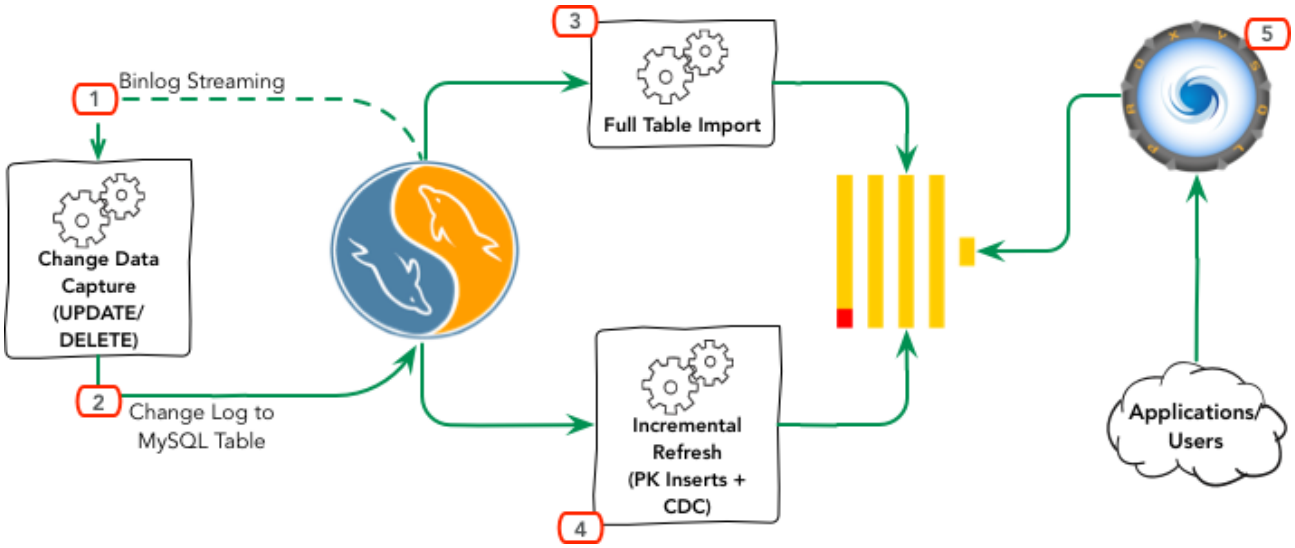

Рис. 1.1. Схема мутації даних ClickHouse

Перший крок — переконатися, що ми фіксуємо зміни з таблиці (таблиць). Тому що технічно ми не можемо визначити, чи відбулося DELETE, якщо це насправді не є INSERT із деякою вказівкою на те, що дані мають бути невидимими. Підтримка цього означає зміну багатьох існуючих таблиць на модель лише INSERT. Тому замість цього ми використовуємо окрему програму для запису змін із двійкових журналів.

Після встановлення доступу до двійкових журналів вихідного сервера MySQL ми фільтруємо зміни, які потрібно записати. У цьому випадку UPDATE і DELETE. Для кожного відповідного зміненого або видаленого рядка ми створюємо запис, який вказує, на який розділ він впливає з відповідної таблиці ClickHouse. З прикладу таблиці вище ми просто перетворюємо стовпець «created_at» у дійсне значення розділу на основі відповідної таблиці ClickHouse.

Перш ніж ми зможемо використати журнал змін, нам доведеться повністю імпортувати нашу таблицю. Загальним випадком використання є простий імпорт із MySQL до ClickHouse із відображенням стовпців один до одного (за винятком, можливо, ключа розділення). В інших випадках нам також доведеться виконувати перетворення стовпців, якщо це необхідно.

Після повного імпорту таблиці та постійного запису журналу змін ми зможемо постійно перебудовувати цю таблицю:

Цей процес перевіряє нашу таблицю журналу змін, щоб визначити, які розділи потрібно оновити. Потім він створив би підмножину цих даних із MySQL, скинув розділ на ClickHouse та імпортував нові дані.

На основі значення PRIMARY KEY з вихідної таблиці MySQL ми також можемо визначити, які нові рядки нам потрібно вивести з вихідної таблиці та вставити в ClickHouse. Якщо розділ, куди входять нові INSERT, уже оновлено, ми пропускаємо цю частину.

1.3.2. Apache Pinot

Apache Pinot є відкритим рішенням для управління базами даних, спеціалізованим на аналітичних запитах в реальному часі. Система розроблена для

високопродуктивної обробки великих обсягів даних і широко використовується в бізнес-аналітиці, моніторингу, рекомендаційних системах та інших сценаріях, де потрібна швидка і точна аналітика.

Однією з ключових особливостей Apache Pinot є його здатність забезпечувати низькі затримки на запити, роблячи його ідеальним вибором для реально-часових аналітичних застосувань. Система підтримує колоночне зберігання даних, що оптимізує читання і значно підвищує швидкість аналітичних запитів.

Pinot є частиною екосистеми проектів Apache і легко інтегрується з іншими популярними технологіями для обробки великих даних, такими як Apache Kafka для потокової обробки даних або Apache Hadoop для зберігання. Система також підтримує розширений набір можливостей, включаючи розподілене зберігання, горизонтальну масштабованість та множення реплік для забезпечення високої доступності.

Спільнота розробників Apache Pinot активно працює над покращенням функціональності та продуктивності системи, і регулярно випускає оновлення, що додатково розширюють можливості для аналітичних застосувань в реальному часі.

В сукупності, Apache Pinot представляє собою високопродуктивну, масштабовану та гнучку систему для реально-часової аналітики, яка може служити ключовим компонентом в сучасних архітектурах обробки великих даних.

Особливості:

- Призначений для аналітики в реальному часі та інтерактивних запитів.
- Масштабована та розподілена архітектура.
- Підтримка комплексної обробки подій.
- Інтеграція з популярними інструментами прийому даних.

Переваги:

- Чудова підтримка аналізу даних у реальному часі.
- Можливість масштабування для роботи з високою швидкістю прийому даних.
- Інтеграція з Apache Kafka для потокової обробки.

Недоліки:

- Менш зрілі порівняно з деякими іншими аналітичними базами даних.
- Для певних випадків використання можуть знадобитися додаткові компоненти.

Розглянемо структуру програмного засобу Apache Pinot більш детально, із використанням діаграми модулів, зображеної на Рис. 1.2.

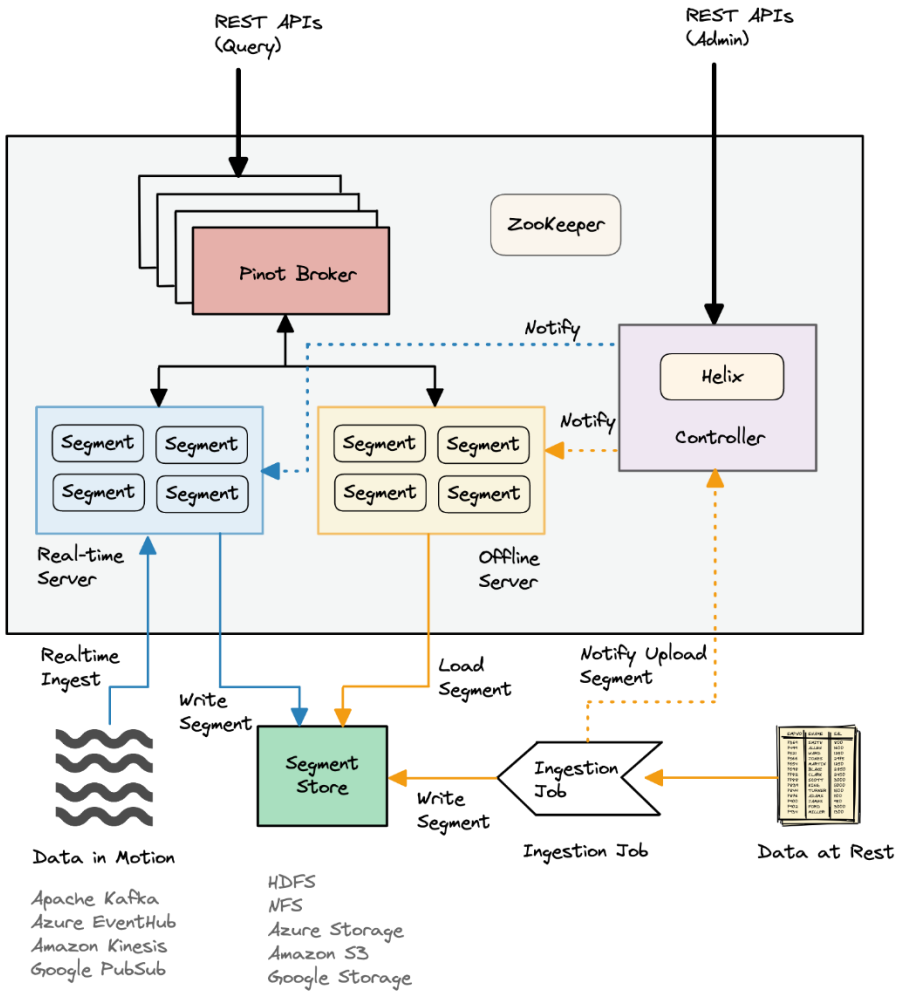

Рис. 1.2. Діаграма компонентів Apache Pinot

Apache Pinot має такі інтегровані компоненти, які використовуються для різних цілей:

- Брокер: обробляє запити Pinot. Брокери приймають запити від клієнтів і паралельно перенаправляють запит на різні сервери, а потім консолідують результати та об’єднують їх у єдину відповідь.

- Контролер: контролер Pinot має різні функції, такі як підтримка різних типів метаданих, таких як конфігурації, схеми тощо. Він керує zookeeper та іншими компонентами Pinot, такими як сервер, брокер, міньйони.
- Сервер: відповідає за прийом даних як у пакетному режимі, так і в реальному часі. Посередник надсилає запити на сервер, а потім збирає результати з сервера.
- Minion: він інтегрований у структуру завдань Helix і відповідає за виконання завдань Pinot і викликає події на основі стану виконання завдання. Можна вказати час виконання завдання за допомогою виразу cron у завданнях-міньйонах.
- Сегмент: Pinot розбиває дані на менші фрагменти/розділи. Їх можна розглядати як часові розділи. Він містить сегменти та розділені дані у вигляді рядків і стовпців.

1.3.3. Apache Druid

Apache Druid — це високопродуктивна, розподілена база даних, спроектована з орієнтацією на аналітику великих об'ємів даних в реальному часі. Система була розроблена для забезпечення швидкого доступу до агрегованих даних і широко використовується в сферах, де потрібна оперативна аналітика, таких як фінанси, реклама, телекомунікації та інтернет-служби.

Одна з ключових особливостей Apache Druid — це здатність обробляти потокові дані на льоту, інтегруючи їх з історичними даними для негайного аналізу. Система використовує колоночне зберігання та індексацію для оптимізації швидкості запитів. Така архітектура дозволяє Druid виконувати складні агрегаційні запити з низькою затримкою.

Apache Druid може масштабуватися горизонтально, що робить його ефективним рішенням для організацій з великими об'ємами даних. Його архітектура підтримує автоматичне балансування навантаження і може протистояти відмовам, завдяки розподіленому зберіганню та обчисленням.

Система інтегрується з іншими популярними інструментами обробки великих даних, такими як Apache Kafka для потокової обробки та Apache Hadoop для пакетної обробки даних. Це робить Apache Druid частиною ширшої екосистеми Big Data, дозволяючи користувачам ефективно розробляти та масштабувати свої аналітичні рішення.

Особливості:

- Оптимізовано для часу відповіді на запит менше секунди.
- Стовпцеве зберігання з підтримкою даних часових рядів.
- Масштабована та розподілена архітектура.
- Можливості прийому даних у реальному часі та пакетної обробки.

Переваги:

- Виняткова продуктивність для аналізу часових рядів даних.
- Підходить для аналітики та моніторингу в реальному часі.
- Гнучкість обробки різних типів даних і випадків використання.

Недоліки:

- Складне встановлення та налаштування.
- Для деяких функцій можуть знадобитися додаткові компоненти.

Apache Kafka відзначається унікальною архітектурою, яка дозволяє будувати потужні системи обміну повідомленнями та обробки потокових даних. Вона складається з брокерів Kafka, які утворюють розподілений кластер, де дані можуть бути розподілені на різні теми та розділені на різні розділи. Це дозволяє обробляти великі потоки даних, забезпечуючи високу продуктивність та масштабованість. Розглянемо архітектуру, що забезпечує такі характеристики задопомоги діаграми виконання розподілених запитів, зображеної на Рис. 1.3.

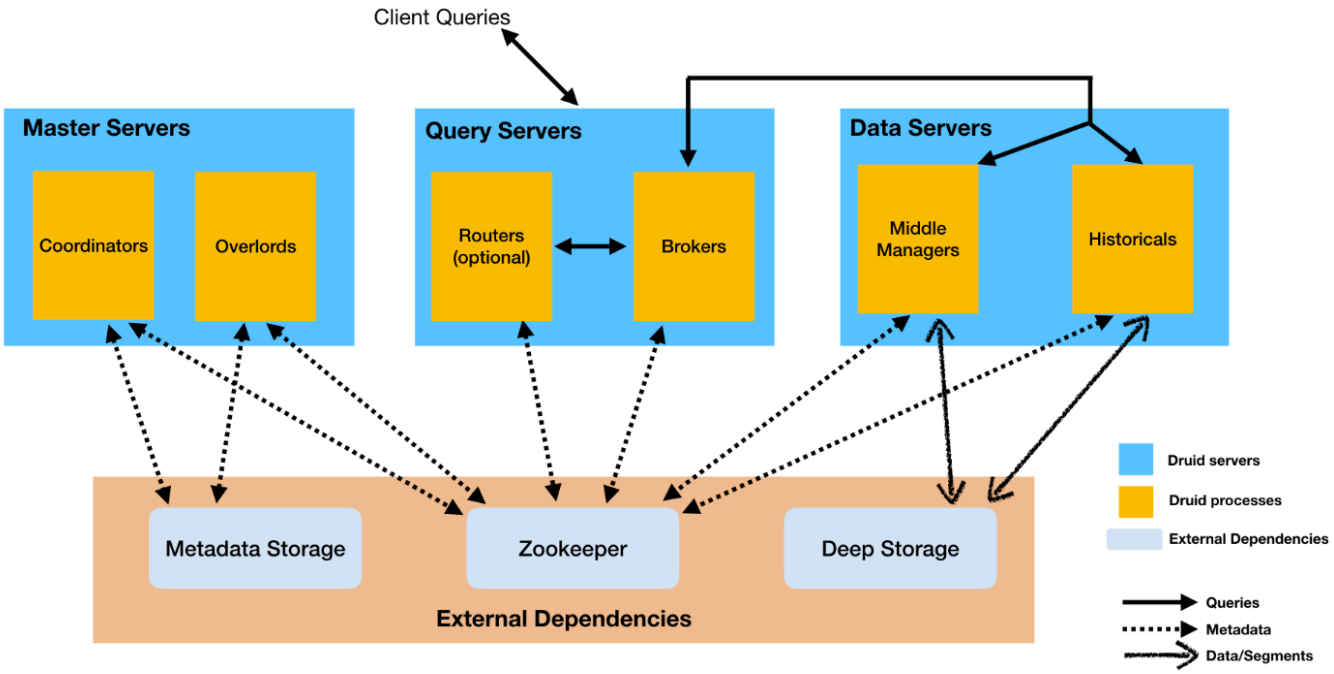

Рис. 1.3. Діаграма виконання розподілених запитів у Apache Kafka

Ахітектура Apache Druid включає в себе кілька ключових компонентів, що взаємодіють для забезпечення ефективної роботи системи:

- Coordinator Service відповідає за доступність даних на кластері. Ця служба забезпечує розподілення даних між різними вузлами Historical Services і оптимізує їх розташування з урахуванням завантаженості, доступності та інших параметрів.
- Overlord Service керує присвоєнням завдань по інгестії даних. Служба відповідає за створення та моніторинг завдань, які забезпечують постійний потік даних в систему. Це включає в себе розподіл завдань між MiddleManager services.
- Broker — це точка входу для зовнішніх клієнтів, які хочуть виконати запити до бази даних. Broker відповідає за маршрутизацію запитів до відповідних Historical або Real-time вузлів для швидкого та ефективного виконання запитів.
- Router Services є додатковим, але необов'язковим компонентом. Ці служби спрямовують вхідні запити до Broker, Coordinator, або Overlord

вузлів. Вони корисні для оптимізації процесу маршрутизації і балансування навантаження.

- Historical Services зберігають дані, які доступні для запитів. Ці вузли є важливою частиною системи, оскільки вони забезпечують швидкий доступ до історичних даних, які були вже проіндексовані та оптимізовані для запитів.
- MiddleManager Services відповідають за процес інгестії даних в систему. Це може включати в себе обробку потокових даних в реальному часі або пакетну обробку даних. MiddleManager взаємодіє з Overlord Service для отримання завдань та їх виконання.

Кожен з цих компонентів відіграє свою унікальну роль в архітектурі Apache Druid, і разом вони забезпечують високу продуктивність, масштабованість та надійність системи. Все це робить Apache Druid одним з передових рішень для аналітики в реальному часі, ідеально підходящим для сценаріїв, що вимагають швидкого доступу до великих та складних наборів даних.

1.4. Техніко-економічна характеристика предметної області

Аналітичні бази даних відіграють ключову роль у сучасному бізнес-середовищі. Вони забезпечують можливість швидкого та ефективного аналізу великих обсягів інформації, що є критично важливим для прийняття обгрунтованих управлінських рішень. Ця характеристика спрямована на оцінку технічних та економічних аспектів аналітичних баз даних.

Технічні аспекти:

- Продуктивність: Аналітичні бази даних оптимізовані для високошвидкісної обробки запитів, часто використовуючи паралельну обробку та спеціалізовані алгоритми.
- Масштабованість: Ці системи можуть легко масштабуватися, щоб обробляти великі обсяги даних.
- Безпека: Захист інформації є важливим питанням, і сучасні рішення зазвичай інкорпорують розширені функції безпеки.

- Інтеграція: Здатність інтегруватися з іншими системами та платформами є однією з ключових переваг.

Економічні аспекти:

- Вартість Розробки та Впровадження: Незважаючи на високі початкові витрати, довгострокова економічна вигода часто переважає вартість.
- ROI (Повернення Інвестицій): Завдяки можливостям аналізу та оптимізації бізнес-процесів, ROI зазвичай є високим.
- Вартість Володіння: Ці системи вимагають постійної підтримки та оновлення, що може додати до загальної вартості володіння.
- Ліцензування та Підписка: Вартість може коливатися в залежності від моделі ліцензування: одноразова покупка, підписка або відкрите програмне забезпечення.

З ростом обсягів даних та попиту на швидке прийняття рішень, ринок аналітичних баз даних продовжує рости. Аналітичні бази даних забезпечують потужні технічні можливості для комплексного аналізу даних. Вони представляють собою високовартісні, але економічно ефективні інвестиційні проекти з великим потенціалом для збільшення ефективності бізнесу. Тому розуміння техніко-економічних аспектів аналітичних баз даних є ключовим для успішного впровадження та експлуатації цих систем.

1.5. Висновки

В даному розділі було проведено всеосяжне дослідження предметної області OLAP (Online Analytical Processing) систем. Розглянуто історичний контекст розвитку аналітичних баз даних, вказано на ключові етапи та дослідження, що вплинули на сучасні підходи в OLAP.

Відзначено важливість багатовимірного аналізу, який є основоположним в OLAP системах. Цей підхід дозволяє забезпечити глибокий і гнучкий аналіз даних, роблячи системи не тільки потужними, але і зручними в користуванні. Також були розглянуті концепції матеріалізації підрахунків та індексації. Ці методики

спрямовані на оптимізацію доступу до даних і можуть значно прискорити обробку запитів.

Техніко-економічна характеристика аналітичних баз даних показала, що впровадження OLAP технологій є не тільки технологічно виправданим, але і економічно ефективним. Сучасні OLAP рішення можуть призвести до значного зростання продуктивності бізнес-аналітики і, як наслідок, до підвищення конкурентоздатності підприємства.

Аналіз існуючих розробок дав можливість оцінити ступінь зрілості ринку OLAP рішень і вказав на те, що попит на високоякісні аналітичні системи росте. Зокрема, було виділено декілька ключових технологій, таких як Apache Druid, ClickHouse і Apache Pinot, які представляють великий інтерес для подальших досліджень.

Отже, можна зробити висновок, що OLAP системи є критично важливою частиною сучасного екосистеми аналітики даних. Вони дозволяють організаціям ефективно управляти великими обсягами інформації, видобувати з неї цінну інформацію і тим самим підвищувати ефективність бізнес-процесів. Сучасні тенденції розвитку OLAP технологій, такі як інтеграція з машинним навчанням, інтернетом речей та іншими передовими технологіями, лише посилюють їх значущість і актуальність для наукових досліджень.

РОЗДІЛ 2
ДОСЛІДЖЕННЯ ВИМОГ ДО СИСТЕМИ

Цей розділ спрямований на дослідження ключових вимог до OLAP систем управління базами даних. Буде розглянемуто ряд факторів, що впливають на вибір OLAP рішень: від технічних аспектів, таких як швидкість обробки запитів, масштабованість і надійність, до функціональних — можливості аналізу, зручність інтерфейсу, інтеграція з іншими системами та інше.

Також буде зроблено спробу надати систематизований погляд на вимоги, що ставляться до сучасних OLAP систем, з оглядом на специфіку різних видів бізнесу і секторів економіки. Розгляд цих питань допоможе не тільки зрозуміти актуальні потреби ринку, але і визначити напрямки для подальших наукових та практичних досліджень в цій області.

В цьому розділі буде розглянуто:

- Технічні вимоги до OLAP баз даних.
- Функціональні вимоги.
- Вимоги з точки зору бізнесу.
- Приклади застоування OLAP систем.
- Постановка задачі на розробку OLAP бази даних.

2.1. Дослідження та класифікація технічних вимог до OLAP баз даних

Далеко не всі OLAP системи створюються однаковими. Для того щоб система могла забезпечити максимальну користь своїм користувачам, вона повинна відповідати ряду строгих технічних вимог. Такі вимоги стосуються як загальної продуктивності та ефективності, так і конкретних аспектів, таких як масштабованість, безпека, сумісність і так далі.

З поглибленням дослідження та впровадженням OLAP систем у різних галузях бізнесу, стає очевидним, що ефективність роботи таких систем залежить не тільки від їх внутрішньої архітектури, але і від здатності відповідати конкретним потребам використання. Наприклад, система, оптимізована для швидкої обробки

невеликих запитів, може не виявитися ефективною при аналізі великих масивів даних. Більше того, сучасні OLAP рішення повинні мати можливість адаптації до динамічно змінюваних бізнес-вимог та технологічних інновацій. Це може включати підтримку нових джерел даних, інтеграцію з новими платформами та сервісами, а також можливість розширення функціоналу за рахунок додаткових модулів або плагінів.

Цей підрозділ призначений для детального аналізу ключових технічних вимог, які пред'являються до сучасних OLAP баз даних. Ми дослідимо те, які конкретні характеристики та можливості необхідні для забезпечення високої продуктивності системи, її надійності та готовності до інтеграції з іншими IT-рішеннями. Розуміння цих вимог допоможе не тільки вибрати оптимальне рішення для конкретних бізнес-задач, але й прокласти шлях для подальшого розвитку та вдосконалення OLAP технологій.

2.1.1. Швидкість обробки запитів

Швидкість обробки запитів є однією з ключових характеристик OLAP баз даних, яка безпосередньо впливає на ефективність аналітичних операцій та задоволеність користувача. Щоб зрозуміти, які вимоги ставляться до швидкості обробки, слід розглянути різні типи запитів та їхні особливості:

Прості запити

Опис: Це базові запити, які не вимагають складного обчислення або аналізу великих обсягів даних.

Вимоги: Відгук системи має бути миттєвим, ідеально в межах мілісекунд до декількох секунд.

Складні аналітичні запити

Опис: Запити, які включають глибокий аналіз даних, такі як знаходження кореляцій, прогнозування або використання статистичних методів.

Вимоги: Час обробки може варіюватися в залежності від складності, але загалом відгук системи повинен бути в межах декількох секунд до декількох хвилин.

Запити в реальному часі

Опис: Запити до даних, які потребують найбільш актуальну інформацію, наприклад, для моніторингу в реальному часі.

Вимоги: Оскільки актуальність даних є критично важливою, час відгуку повинен бути миттєвим, зазвичай в межах мілісекунд.

Масові запити

Опис: Запити до великих обсягів даних, наприклад, коли потрібно отримати зведені дані за рік чи декілька років.

Вимоги: Час обробки може бути довшим, але все ще повинен бути прийнятним для користувача, зазвичай в межах декількох хвилин.

Інтерактивні запити

Опис: Запити, які виникають під час інтерактивної роботи користувача з OLAP системою, наприклад, під час візуалізації даних.

Вимоги: Оскільки користувач очікує миттєвої відповіді під час інтеракції, час відгуку повинен бути максимально швидким, в межах мілісекунд до секунд.

Враховуючи цю класифікацію, можна краще зрозуміти, які вимоги пред'являються до швидкості обробки запитів в OLAP системах, і які аспекти оптимізації можуть бути застосовані для кожного типу запиту.

2.1.2. Масштабованість

Масштабованість є однією з ключових технічних характеристик OLAP систем, яка дозволяє оптимально використовувати ресурси та адаптуватися до збільшення обсягу даних або кількості користувачів.

Реплікація

Опис: Реплікація дозволяє створювати копії даних на різних вузлах або серверах, забезпечуючи високу доступність та знижуючи ризик втрати даних.

Вимоги:

- Автоматичне відновлення: У випадку відмови одного з вузлів система повинна автоматично переключитися на робочий вузол без втрати даних або продуктивності.

- Консистентність: Всі репліки повинні бути актуалізовані в реальному часі, забезпечуючи консистентність даних по всіх репліках.

Шардинг

Опис: Шардинг дозволяє розподіляти датасет між різними серверами чи вузлами, забезпечуючи оптимальний розподіл навантаження та ефективне зберігання.

Вимоги:

- Гнучкість: Система повинна мати змогу динамічно реалокувати шарди з урахуванням зростаючого обсягу даних.
- Балансування навантаження: Навантаження між вузлами повинно рівномірно розподілятися, забезпечуючи оптимальний робочий режим кожного вузла.
- Прозорість для користувача: Процес шардингу має бути прозорим для користувача, гарантуючи, що запити до бази даних обробляються без затримок, незалежно від того, на якому вузлі зберігається конкретний шард.

2.1.3. Консистентність даних

Персистивність в OLAP системах відноситься до довготривалого зберігання даних, гарантуючи їхню цілісність, доступність та безпеку.

Опис: Забезпечення надійного, безпечного та довготривалого зберігання даних з можливістю відновлення в разі втрат.

Вимоги:

- Відновлення даних: У випадку втрати даних (через відмову обладнання, помилки користувача тощо) система повинна мати механізми для їх відновлення.
- Резервне копіювання: Система повинна забезпечувати автоматичне резервне копіювання даних з можливістю налаштування періодичності та параметрів зберігання.

- Транзакційна консистентність: Забезпечення консистентності даних при паралельній обробці транзакцій.

У разі несподіваного завершення роботи, база даних повинна мати механізми відновлення незавершенних операцій над даними.

З урахуванням цих вимог, можна розуміти, що масштабованість та персистивність є вітальними компонентами будь-якої ефективної OLAP системи, і їх реалізація потребує глибокого технічного розуміння та уваги до деталей.

2.2. Постановка задачі

За допомогою попереднього аналізу предметної області вдалося з'ясувати основні недоліки та переваги наявних програмних рішень. Тепер будуть виділені основні напрямки, задачи в яких повинна брати на себе сучасна система управління анаілтичними базами даних.

2.2.1. Першорядні задачі системи

Серед першорядних задач можна виділити:

- Збереження даних у таблицях. Основа будь-якої бази даних - це її здатність зберігати даних у структурованому форматі. Для аналітичних баз даних, де основний акцент ставиться на обробку та аналіз, ефективне зберігання даних у таблицях є ключовим. Таблиці забезпечують логічний, систематизований спосіб організації даних, що спрощує доступ та операції з ними.
- Виконання класичних запитів до таблиць. Класичні запити, такі як SELECT, INSERT, UPDATE та DELETE, є основними операціями, які дозволяють інтерактивно роботу з даними в базі. Вони забезпечують можливість зчитування, вставки, оновлення та видалення даних, формуючи базовий функціонал взаємодії користувача з даними.
- Виконання агрегації по стовпцям з ціллю аналізу даних. Агрегація даних є однією з основних операцій в аналітичних запитах. Це дозволяє користувачам отримувати узагальнені дані, такі як суми, середні

значення, максимальні та мінімальні значення по конкретним стовпцям, що полегшує аналіз та виведення висновків.

- Виконання запитів до материалізованих даних (slice-and-dice запити). Slice-and-dice запити дозволяють користувачам аналізувати підмножини даних з різних точок зору. Завдяки материалізованим виглядам, які зберігають попередньо обчислені результати, ці запити виконуються набагато швидше, надаючи можливість користувачам "різати" та "кидати кубики" даних з високою продуктивністю.
- Фільтрація результатів запитів. Фільтрація є ключовою операцією при обробці запитів, яка дозволяє відсіювати неважливі або неактуальні дані. Завдяки ефективним механізмам фільтрації, користувачі можуть зосередитися на конкретних даних.

2.2.2. Другорядні задачі системи

Серед другорядних задач можна виділити:

- Зберігання даних по стовпцям. Зберігання даних в аналітичних базах даних за стовпчиковим принципом оптимізує доступ до даних, який відбувається на рівні окремих стовпців. Це поліпшує продуктивність запитів, які сканують або агрегують обмежені обсяги стовпців, а також дозволяє ефективніше стискати дані.
- Стиснення даних. Для ефективного використання сховища та забезпечення швидкодії запитів, аналітичні бази даних використовують різні техніки стиснення даних. Це дозволяє значно зекономити місце зберігання та прискорює обробку запитів.
- Реплікація даних та обробка запитів декількома серверами одночасно. Реплікація забезпечує високу доступність та надійність системи. Аналітичні бази даних можуть розподіляти навантаження запитів між декількома серверами, що підвищує продуктивність та стабільність системи в цілому.

- Індексація даних. Індексація – це створення спеціальних структур даних, що спрощують доступ до даних в базі. Використовуючи індекси, система може швидко знаходити потрібні записи без необхідності сканування всієї таблиці.
- Наближені розрахунки. Великі аналітичні бази даних можуть використовувати наближені алгоритми для швидкого виконання складних обчислень, жертвуючи невеликою точністю відповіді заради швидкості. Це особливо корисно для великих датасетів, де абсолютна точність може бути менш важливою, ніж швидкість отримання результату.

2.3. Огляд функціональних моделей системи

Для того, щоб ознайомитися з запропонованим варіантом вирішення задачі, в рамках дипломного проекту були розроблені функціональні моделі даних IDEF0 "TO-BE" (тобто така функціональна модель даних відображає структуру і зв'язок завдань в такому вигляді, в якому це повинно бути реалізовано), IDEF0 описує конкретні кроки процесу та зв’язки, що склалися. Він також записує інформаційні потоки, що виникають внаслідок цих взаємозв’язків.

2.3.1. Збереження даних у базі

Розглянемо фукнціональну модель даних процезу збереження даних у базі, зображену на Рис. 2.1. Рисунок 2.1 є важливим з огляду на його візуальне відображення процесу збереження даних у таблиці під час первинної конфігурації. Цей рисунок включає схему таблиці, яка визначає структуру даних, призначених для збереження, а також самі дані, що будуть занесені в таблицю. Це візуальне представлення передає етапи валідації схеми таблиці, створення самої таблиці та процес збереження даних. Така діаграма спрощує розуміння процесу, оскільки вона вказує на основні кроки та акторів, що беруть участь у збереженні даних в базі даних.

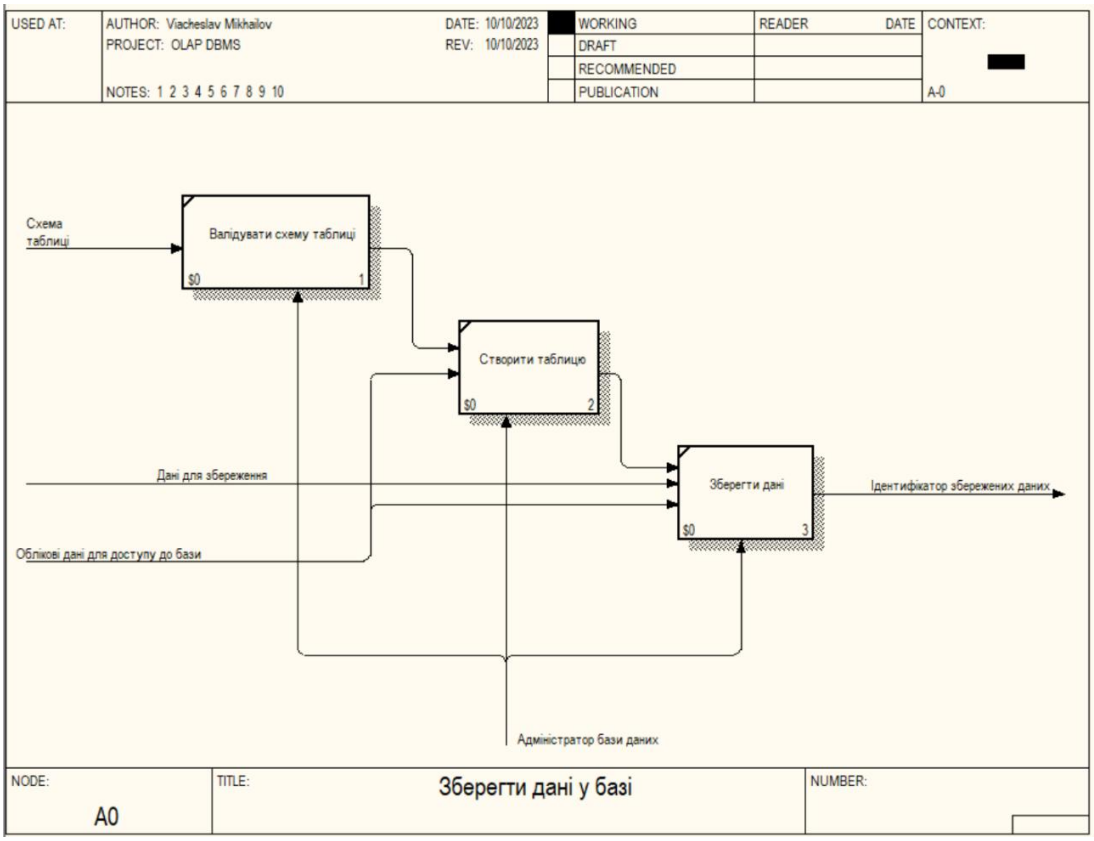

Рис. 2.1. Функціональна модель даних IDEF0 (TO-BE) процесу збереження даних у базі, рівень A0

На рисунку 2.1 зображено переміщення даних в процесі первинної конфігурації таблиці та збереження даних у ній.

Вхідні дані:

- Схема таблиці: Опис структури таблиці, до якої будуть додані дані.
- Дані для збереження: Конкретні дані, які потрібно зберегти в таблиці.
- Загальний доступ до бази: Інформація про доступ до бази даних.

Функції (або завдання):

- Валідувати схему таблиці: Визначення структури таблиці для збереження даних.
- Створити таблицю: Ініціація процесу створення таблиці на основі обраної схеми.
- Зберегти дані: Процес внесення даних у створену таблицю.

Вихідні дані:

- Ідентифікатор збережених даних: Унікальний код або номер, який ідентифікує записи, які були додані в таблицю.

Актори:

- Адміністратор бази даних: Відповідальна особа або система, яка здійснює контроль та управління процесом збереження даних в базі.

Ця діаграма допомагає візуалізувати та зрозуміти, які кроки виконуються при зберіганні даних у базі та які актори беруть участь у цьому.

2.3.2. Виконання аналітичного запросу із використанням агрегації

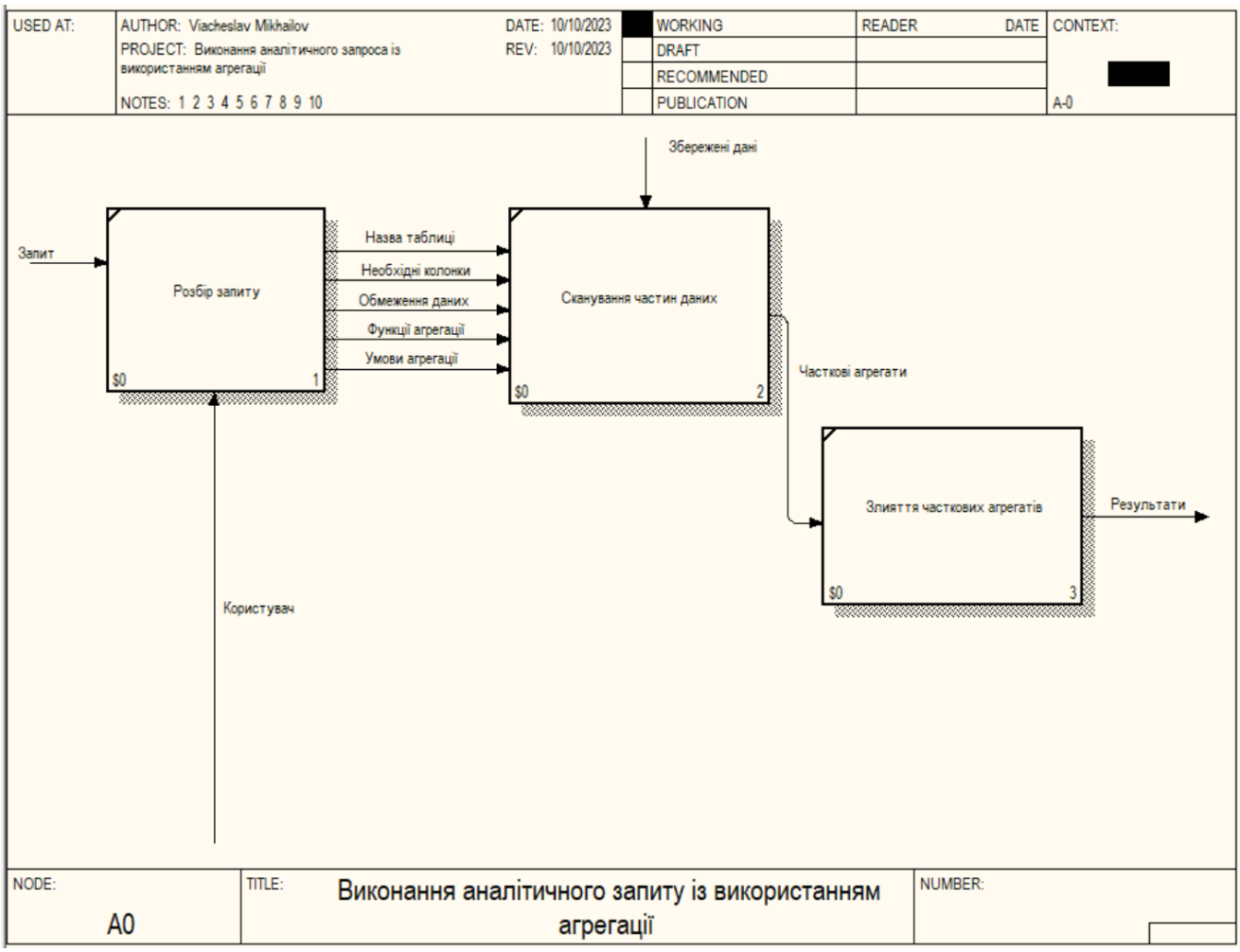

Рис. 2.2. Функціональна модель даних IDEF0 (TO-BE) процесу виконання запиту із використання агрегації, рівень A0

На рисунку 2.2 представлено бізнес-процес "Виконання аналітичного запиту із використанням агрегації" в OLAP базі даних.

Вхідні дані цього процесу - сам аналітичний запит, який потребує обробки. Процес включає етапи розбору запиту для визначення даних, які потрібно взяти для обробки, а також злиття цих частин даних у єдину структуру для подальшої агрегації.:

- Запит: Це сам аналітичний запит, який потребує обробки.

Послідовність дій:

- Розбір запиту. Перший крок полягає у розборі запиту, де система розуміє, які дані потрібно отримати і які операції з ними виконати.
- Складання частини даних. Після розбору запиту система визначає, які частини даних потрібно взяти для обробки.
- Злиття часткових агрегатів. Всі частини даних, які були зібрані на попередньому етапі, об'єднуються в одну структуру даних.

Вихідні дані цього процесу - фінальний результат аналітичного запиту, який представляє відповідь на запит з використанням агрегації. Деталі запиту, такі як назва таблиці, необхідні колонки, обмеження даних, функції та умови агрегації, використовуються під час розбору запиту для уточнення його деталей:

- Результат: Фінальний результат аналітичного запиту, який відображає відповідь на запит з використанням агрегації.

Додаткова інформація: Назва таблиці, Необхідні колонки, Обмеження даних, Функції агрегації, Умови агрегації: Це додаткова інформація, яка використовується під час розбору запиту для з'ясування деталей запиту.

Цей бізнес-процес відображає послідовність дій, які відбуваються в OLAP базі даних при обробці аналітичного запиту з агрегацією. А діаграма ілюструє важливий етап обробки даних у OLAP базі даних, допомагаючи краще зрозуміти послідовність дій та процес опрацювання аналітичних запитів з використанням агрегації.

2.3.3. Виконання slice-and-dice запиту

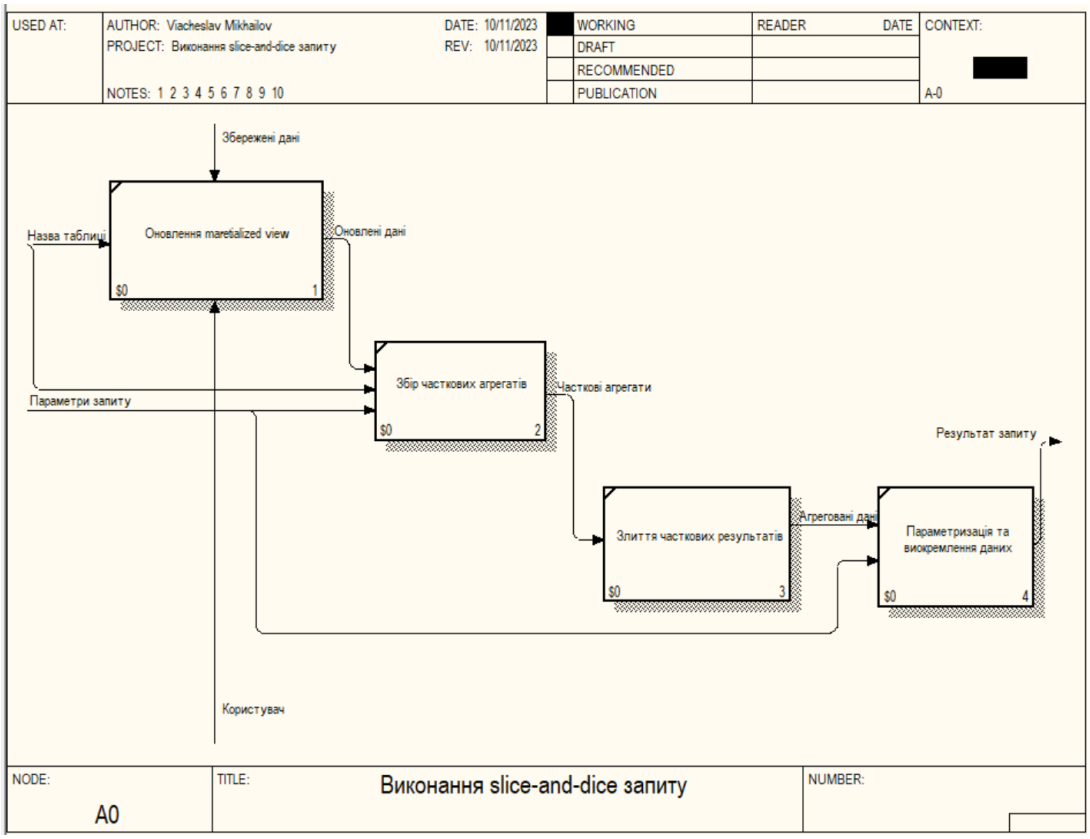

Рис. 2.3. Функціональна модель даних IDEF0 (TO-BE) процесу виконання запиту до материалізованого зрізу даних, рівень A0

На рисунку 2.3 зображений процес планування резервного копіювання файлу адміністратором. При цьому система вимагає введення опцій правила, які дозволяють гнучко сконфігурувати графік та сам процес.

Можна відзначити, що на виході користувач (адміністратор) отримує всі дані про сконфігуроване правило.

Дану модель можна декомпозувати ще на кілька рівнів з 3 або 4 моделями в кожному. Однак діаграма IDEF0 передбачає лише визначення функціональної моделі даних процесів щодо користувача, в той час як програмний комплекс, що розробляється, більшою мірою спирається на процеси, приховані від користувача, наприклад, ось деякі з них:

- Фонові процеси, що переглядають графік резервного копіювання, та додають потрібні файлі до черги на копіювання коли це необхідно

- Фонові процеси очищення сховищ від файлів, що більше не знаходяться під контролем системи(коли правила з їх резервного копіювання було деактивовано або видалено)

Тому такі процеси системи у розділі, присвяченому проектуванню, буде розглянуто задопомоги відповідних UML-діаграм.

2.4. Загальна характеристика організації вирішення задачі

Очевидно, що при розробці будь-якого рішення з автоматизації необхідно максимально змістовно і конкретно відповісти на ряд питань, або, якщо говорити технічною мовою, попередньо визначити відповідні вимоги до рішення, що проектується і розробляється, яким у рамках даного дипломного проекту виступає OLAP СУБД.

2.4.1. Перелік вимог до системи

На основі згаданої раніше інформації, розглянемо ряд основних вимог до розроблюваної інформаційної системи в рамках даної роботи:

1. Ступінь автоматизації. Вимагається високий рівень автоматизації для OLAP системи, що забезпечує здатність автоматично обробляти великі обсяги даних і різні види аналітичних запитів, мінімізуючи потребу в ручному втручанні.
2. Архітектура. OLAP система повинна використовувати многорівневу архітектуру, що дозволить їй ефективно розподіляти ресурси, адаптуватися до різних обсягів даних та оптимізувати відгук на різні типи запитів.
3. Масштаб. Система повинна бути масштабованою, здатною підтримувати роботу з різними обсягами даних, від малих до дуже великих, та забезпечувати стабільність при великій кількості одночасних користувачів. Система повинна передбачати використання великими компаніями із великими обсягами даних.

4. Функціональність. Очікується, що система надасть користувачам широкий спектр аналітичних інструментів, таких як агрегація, багатовимірний аналіз, фільтрація, сортування, і так далі.
5. Зручність використання. Інтерфейс системи повинен бути інтуїтивно зрозумілим та дружнім до користувача, із набором зручних інструментів для створення запитів та візуалізації результатів.
6. Безпека. Вимагається висока безпека даних і аудиту, з можливістю регулювання доступу на різних рівнях, шифруванням даних та захистом від несанкціонованого доступу. Система повинна відповідати сучасним вимогам безпеки - використовувати зашифроване з'єднання між клієнтом та сервером, засноване на алгоритмах асиметричного і симетричного шифрування. Обробка особистої інформації користувачів системи повинна проводитися відповідно до GDPR.
7. Продуктивність. Оптимальна продуктивність є ключовою, і система повинна гарантувати швидкі відгуки на запити, незалежно від обсягу даних або складності запиту.
8. Підтримуваність. OLAP система повинна бути легко підтримуваною, із можливістю швидкого внесення змін, розширення функціоналу та інтеграції з іншими системами без великих витрат часу та ресурсів.
9. Обмеження. Необхідно чітко визначити обмеження системи, зокрема в плані обробки даних, швидкості відповіді на запити та можливих форматів входу/виходу.

В розробці OLAP системи управління базами даних важливо враховувати всі вищезазначені вимоги, оскільки вони визначають успішність і ефективність системи в реальних умовах використання. Це забезпечує високий рівень задоволеності користувача і допомагає досягти поставлених бізнес-цілей.

2.4.2. Технічне завдання

Для більш зручного і точного сприйняття функціоналу та ряду бізнес-правил було створено технічне завдання до розроблюваної інформаційної системи.

Технічне завдання є основним документом, відповідно до якого ведеться розробка інформаційної системи і подальше її приймання при введенні в експлуатацію. Технічне завдання дозволяє однозначно трактувати замовлення і усунути можливе нерозуміння між замовником і виконавцем.

1. Загальні відомості.
   1.1. Найменування системи.
      1.1.1. Повне найменування: OLAP система управління базами даних.
      1.1.2. Скорочене найменування системи: OLAP СУБД.
   1.2. Порядок оформлення і надання замовнику результатів робіт.
      1.2.1. Роботи зі створення програмного комплексу здаються Розробником поетапно в суворій відповідності до чинного календарним планом проекту, на кожному з етапів Замовнику надаються відповідні документи, зазначені в договорі.
2. Призначення і цілі створення системи
   2.1. Призначення системи
      2.1.1. OLAP СУБД призначена для зниження кількості ресурсів, що витрачаються всередині компаній на впровадження, супровід і повсякденне використання механізмів зберігання, передачі і аналізу даних. В рамках проекту автоматизується діяльність в наступних бізнес-процесах:
         - Конфігурація бази даних.
         - Створення таблиць у базі.
         - Збереження даних у таблицях.
         - Зчитування даних із таблиць.
         - Агрегація даних по стовпчикам.
         - Аналіз даних із використанням материалізації підрахунків.
         - Аналіз даних із використанням роздільного аналізу.
         - Резервне копіювання даних.
   2.2. Мета створення системи. Аналітична база даних створюється з метою:

- Підтримки прийняття рішень. Однією з основних причин створення аналітичних баз даних є необхідність у високоякісному і швидкому інструменті для підтримки прийняття рішень на різних рівнях управління. Аналітичні бази даних дозволяють виводити з даних корисні висновки та стратегії.
- Виявлення шаблонів та тенденцій. Дані можуть містити приховані шаблони та тенденції, які не завжди очевидні на перший погляд. Аналітичні бази даних дозволяють ефективно аналізувати великі обсяги даних і виявляти ці шаблони, надаючи цінні відомості для бізнесу.
- Підвищення ефективності бізнес-процесів. Дані можуть містити приховані шаблони та тенденції, які не завжди очевидні на перший погляд. Аналітичні бази даних дозволяють ефективно аналізувати великі обсяги даних і виявляти ці шаблони, надаючи цінні відомості для бізнесу.
- Персоналізації взаємодії з клієнтами. Аналіз даних може допомогти організаціям краще розуміти своїх клієнтів і надавати їм персоналізовані пропозиції та рекомендації, що підвищує лояльність та задоволеність клієнтів.
- Прогнозування та планування. На основі історичних даних та актуальних трендів, аналітичні бази даних можуть надавати засоби для прогнозування майбутніх подій. Це може бути корисно для планування виробничих потужностей, запасів, маркетингових кампаній та інших бізнес-ініціатив.

3. Вимоги предметної області

3.1. Розгортання серверу бази даних можливо на пристроях Linux.

3.2. База даних повинна підтримувати кластеризацію, тобто створення бази на основі декількох серверів, дані та нагрузка між якими розподіляються.

3.3. База даних повинна будуватися на сучасних та ефективних алгоритмах передачі, збереження та аналізу даних, таких як: стовпчикове зберігання даних, індексація даних на основі первинного ключа та зрізів, послідовний запис даних на основі алгоритму злиття, розподілена агрегація даних.

3.4. Специфікація даних. Залежно від предметної області бізнесу, може виникнути потреба в обробці певного типу даних: часові ряди, географічні даних, текстових даних тощо. Аналітична база даних повинна підтримувати такі типи даних та відповідні функції обробки.

3.5. Семантична консистентність. Предметні області часто мають свої терміни та поняття. Аналітична база даних повинна забезпечувати їх узгоджене використання, щоб уникнути непорозумінь або невірного інтерпретування даних.

3.6. Гнучкість запитів. Деякі предметні області можуть мати унікальні запити до аналізу даних. Система повинна бути гнучкою і дозволити користувачам формувати складні запити, які враховують специфіку предметної області.

3.7. Інтеграція з іншими системами. Часто дані для аналітичної бази даних надходять з різних джерел, які можуть використовуватися в предметній області. Аналітична база даних повинна мати можливість інтеграції з цими системами для автоматичного збору, оновлення та обробки даних.

3.8. Відповідність законодавчим вимогам. У деяких предметних областях, таких як медицина або фінанси, існують строгі законодавчі вимоги до зберігання та обробки даних. Аналітична база даних повинна відповідати цим вимогам, забезпечуючи конфіденційність, цілісність та доступність даних.

2.5. Висновки

Аналіз вимог є основною частиною розробки, на яку надалі спираються команди розробників. У цьому розділі були детально розглянуті всі вимоги, висунуті до системи, або, інакше - всі функції, якими повинен володіти проектований програмний комплекс.

В ході аналізу вимог до системи вдалося максимально абстрагуватися як від більш високого рівня - глобальних завдань, які має вирішувати програмне забезпечення (описаних в розділі 1), так і від наступного, більш низького рівня - подробиць імплементації як призначених для користувача інтерфейсів, так і компонентної архітектури.

Далі будуть декомпонизовані розглянуті в даному розділі вимоги, і виявлені подробиці реалізації кодової бази, яка задовольняє всім їм. Даний процес відбудеться під контролем оформленого технічного завдання, що дасть чітке уявлення про систему, яка повинна вийти в результаті.

РОЗДІЛ 3
ДОСЛІДЖЕННЯ АЛГОРИТМІВ. МЕТОДИЧНІ ВКАЗІВКИ З ВИБОРУ ТА РЕАЛІЗАЦІЇ АЛГОРИТМІВ

У минулих розділах дипломної роботи було досліджено вимоги до сучасних систем управління базами даних, що використовуються для аналізу даних, а також історичні і бізнес чинники, що сприяли виникненню таких вимог.

Якби ціллю цього проекту було дослідження звичайних транзакційних баз даних – минулі розділи залишилися б дуже схожими, адже будь яка база даних в цілому виконує ту саму задачу – збереження та надання доступу до даних, часто із ціллю їх аналізу. Звичайні бази можуть виконати частину із досліджених вимог, проте чи впорається, наприклад, MSSQL Server із аналітичнім записом з агрегацією таблиці розміром 5 терабайт? Чи зможе він взагалі зберігати стільки даних? Як багато часу це вимагає? Чи можна скоротити цей час паралельною обробкою? Якщо ні, то які саме процеси є вузьким місцем, та вимагають оптимізації? Саме на дослідження та пошук вирішення цих та схожих питанньі спрямований цей розділ.

Із метою дослідження засобів, що дозволяють сучасним аналітичним базам даних виконувати складні запити до терабайт даних у реальному часі, цей розділ включає дослідження таких питань:

- порівняння B-дерев та LSM-дерев як засобів первинної індексації даних з точки зору аналітичних баз даних
- необхідність стовпчикового зберігання даних у аналітичних базах даних
- секціонування або розподіл даних між серверами із метою горизонтального масштабування
- використання та засоби реалізації реплікації даних із метою зберігання даних географічно ближче до користувача, підвищення відмовостійкості та горизонтального масштабування нагрузки при читанні даних

- необхідність у транзакціях, переваги відмови від транзакцій для аналітичних баз даних

3.1. Алгоритми засобв первинної індексації даних. Бінарні та журнальовані індекси

Індекс є невід'ємною частиною систем управління базами даних від часів їх появлення, адже він відіграє ключову роль у ефективності пошуку необхідних даних. Загалом, індрекс – це структура даних, що зберігає деякі метадані, що є дорожньою картою при пошуку потрібних даних.

В той самий час, індексация створює певну додаткову навантаженність на операції додавання або оновлення даних, що призводить до пошуку компромісу між швидкістями запису та читання даних, і пріоритет віддається тим операціям, що є найчастішими при використянні для конкретної задачі.

3.1.1. Індекси на основі бінарних дерев

Ряд факторів, таких як: потреба у гнучкості при пошуку компромісу між швидкостями читання та запису, відсутність завеликих обсягів даних, особливості реалізації файлових систем та апаратних носіїв інформації, та інших; історично склалося, що саме бінарні дерева почали використовуватися у 1970 роках [1] та наразі є найпоширенішим видом індексації.

B-дерева - це структура даних, яка використовується для розподілу бази даних на блоки або сторінки фіксованого розміру, зазвичай 4 кілобайти (іноді більше). Читання та запис відбуваються по одній сторінці за раз. Цей підхід є оптимальним для апаратного забезпечення, оскільки диски також розділяються на блоки фіксованого розміру. Кожна сторінка має власну адресу або місцезнаходження, завдяки чому одні сторінки можуть посилатися на інші - схоже на вказівники, але на диску, а не в пам'яті. Цими посиланнями на сторінки можна скласти дерево сторінок.

Одна зі сторінок призначається коренем B-дерева, з якого починається будь-який пошук ключа в індексі. Ця сторінка містить кілька ключів та посилання на дочірні сторінки. Кожна з цих сторінок відповідає за певний діапазон ключів, а

ключі між посиланнями вказують на розташування меж цих діапазонів. Розподіл на сторінки зображено на Рис. 3.1.

При цьому, пошук по такому індексу виконується із використанням алгоритму бінарного пошуку, тобто має складність O(n log n).

3.1.2. Індекси на основі журнальованих дерев злияття

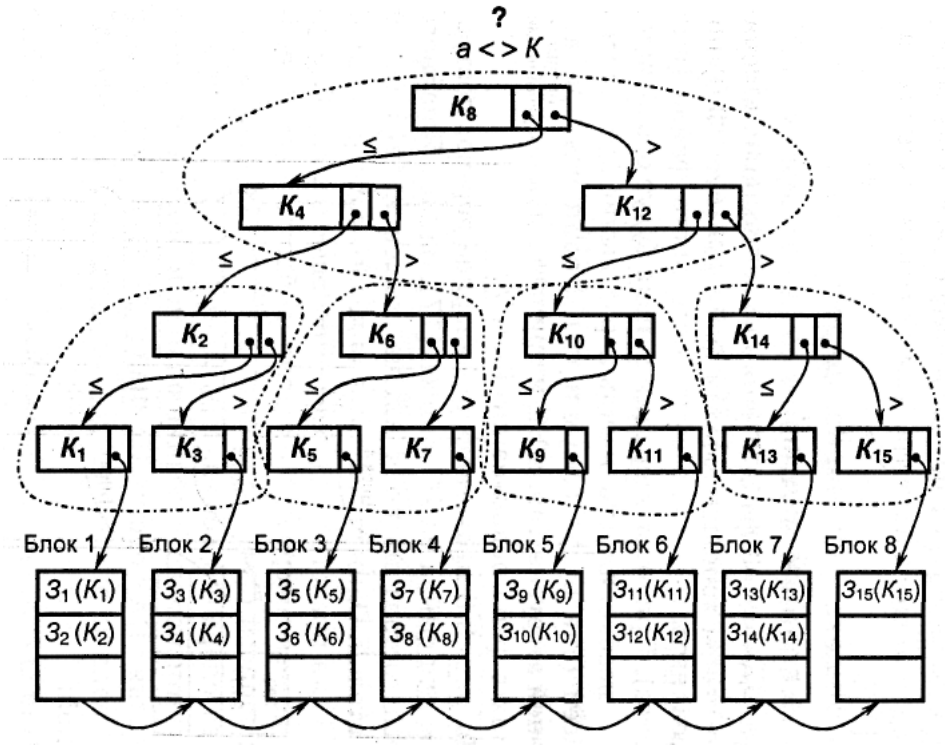

Рис. 3.1. Розподіл даних на сторінки за схемою збереження в індексі на основі бінарного дерева

Журнальовані індекси (LSM дерева) мають простішу ідею, проте мають і свої недоліки, саме тому вони тільки набирають свою популярність в порівнянні із індексами на основі бінарних дерев. Дані у журнальованому індексі зберігаються послідовно, що чергу дозволяє значно пришвидшити запис даних на диск по декільком причинам:

- послідовний запис на жорсткі диски має значно більшу швидкість. хоча ця перевага частково нівелюється сучасними твердотільними накопичувачами, послідовний запис все ще надає перевагу [2].
- відсутнє дублювання даних до таблиці упередженого запису.
- відсутній перезапи даних під час запису нових строк у випадку розбиття сторінок бінарного дерева.

Окрім того, дані зберігаються по частинам, що дозволяє ефективно стискати дані, а також значно облегшує процеси реплікації та секціонування даних між декулькома серверами із метою розподілення нагрузки та обсягів даних.

Проте, журнальований індекс не програє, а у деяких випадках навість виграє у швидкості зчитування даних за рахунок зберігання даних у відсортованому вигляді. Саме тому цей тип індексу містить у назві слова “дерево злияття”, адже відсортованість досягається завдяки використання алгоритму сортування злиттям та дерев, що самобалансуються (красно-червоні або AVL дерева):

- при додаванні нового запису, спочатку він записується не на диск, а до AVL-дерева в оперативній пам'яті, що зазвичай називається MemTable (таблиця, що знаходиться у пам'яті).
- коли дерево у пам'яті перевищує за розміром декілька мегабайтів – воно записується на диск у окремий сегмент у відсортованому вигляді. Такий запис є високоефективним, адже дані вже відсортовані завдяки двикористанню дерева.
- при виконанні запиту на читання даних, спочатку пошук відбувається у дереві в опреативній пам'яті, а потиі – у сегментах на диску.
- час від часу у фоновому процесі запускаються операції злияття пар сегментів. Злияття також є високоєфективним процесом, адже дані вже є відсортованими, тож цей процес має складність O(n). Процес злияття зображено на Рис. 3.2.

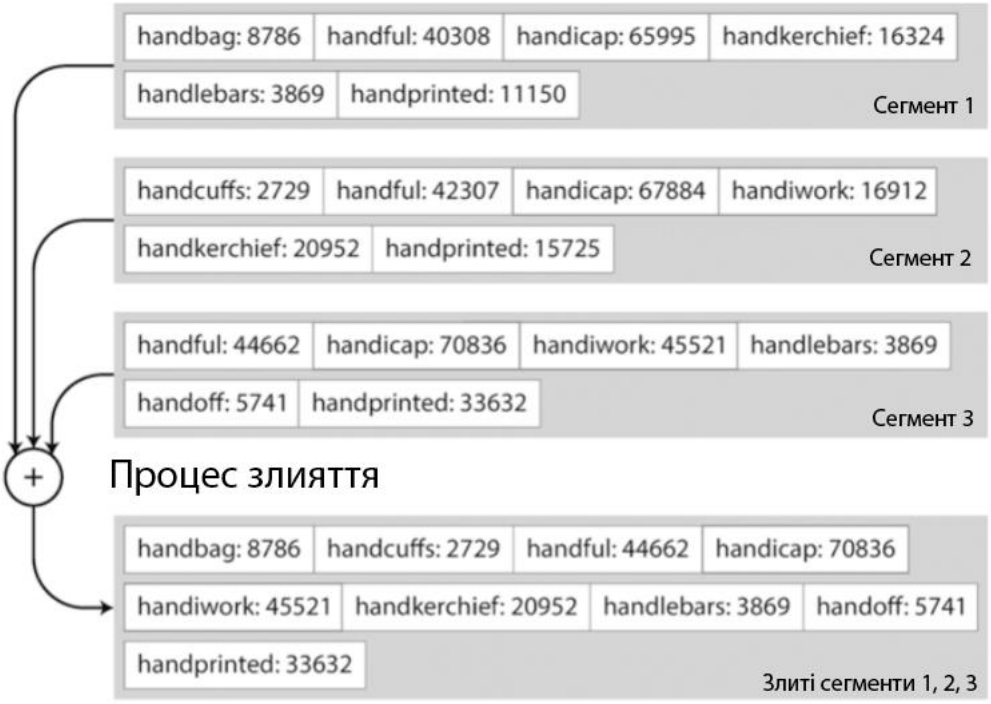

Рис. 3.2. Процес злиття сегментів даних у журнальованому дереві

Окрім того, використання журнальованого індексу передбачає зберігання метаданих у вигляді байтового зміщення у сегменті строки із відповідним ключем, як зображено на Рис. 3.3.

Тобто, потрібний запис завжди може бути знайдений швидко із

Рис. 3.3. Структура індексу журнальованого дерева

використанням бінарного пошуку.

3.1.3. Порівняння властивостей

Таблиця 3.1.

Властивості бінарних дерев та журнальованого індекса

№	Властивість	Бінарне дерево	Журнальований індекс
1	Запис даних	Безпосередньо в структуру дерева, що може призвести до частого перезаписування і фрагментації даних.	Швидкі записи в оперативній пам'яті, з подальшим об'єднанням в консолідовані файли для оптимізації записів.
2	Читання даних	Зручний доступ до даних через ієрархічну структуру дерева, що забезпечує ефективний пошук.	Читання може бути повільним для випадкового доступу до даних, оскільки вимагає послідовного злиття та компактування записів.
3	Компактність	Менш компактний, бо кожна сторінка може містити незадіяні ділянки простору, що призводить до фрагментації.	Для оптимізації використовується злиття сторінок, тому може бути більш компактним завдяки збереженню даних в консолідованих файлах, які ефективно використовують простір.
4	Транзакційність	Можливе досягнення сильно вираженої транзакційності, так як блокування диапазону ключів може бути прив'язане до дерева [3].	Майне недосяжна, адже журнальований індекс не передбачає наявність таблиці упередженого запису, а також через низку інших питань, таких як можливе стискання частини даних, що використовується в транзакції, якщо транзакція триває занадто довго [4]

3.1.3. Порівняння продуктивності

Загалом існує три критичні показники для вимірювання продуктивності структури даних: посилення запису, посилення читання та посилення простору.

Для жорстких дисків (HDD) вартість перемикання між секціями диска є великою, тому продуктивність довільного читання/запису гірша, ніж продуктивність послідовного читання/запису. У цій статті передбачається, що використовується флеш-пам’ять, тому ми можемо ігнорувати вартість перемикання між секціями диска.

Таблиця 3.2.

Опис метрик продуктивності баз даних

№	Метрика	Опис
1	Посилення запису	Посилення запису — це відношення кількості даних, записаних на запам’ятовуючий пристрій, до кількості даних, записаних у базу даних. Наприклад, якщо при запису 10 МБ до бази даних спостерігається швидкість запису на диск 30 МБ, посилення запису дорівнює 3.
2	Посилення читання	Посилення читання — це кількість читань диска на запит. Наприклад, якщо базі даних потрібно прочитати 5 сторінок, щоб відповісти на запит, підсилення читання дорівнює 5.
3	Посилення простору	Посилення простору – це співвідношення обсягу даних на пристрої зберігання даних до обсягу даних у базі даних. Наприклад, якщо розміщується 10 МБ у базі даних, і ця база даних використовує 100 МБ на диску, тоді посилення простору дорівнює 10.

У бінарному дереві копії ключів зберігаються у внутрішніх вузлах; ключі та записи зберігаються у листах; крім того, кінцевий вузол може містити вказівник на наступний кінцевий вузол для підвищення продуктивності послідовного доступу. Для спрощення аналізу припустимо, що розмір блоку дерева дорівнює B та вимірюється в байтах, а ключі, покажчики та записи мають постійний розмір, тому кожен внутрішній вузол містить O(B) дітей і кожен листок містить O(B) записів даних. За всіма цими припущеннями глибина дерева дорівнює $O(\log_B N/B)$, де N – розмір бази даних. Для найгіршого випадку робочого навантаження вставки кожна вставка вимагає запису кінцевого блоку, що містить запис, тому посилення запису дорівнює B. А кількість читань диска на запит становить не більше $O(\log_B N/B)$ що є глибиною дерева.

У той час, у журнальованому індексі на основі рівнів, дані організовані за рівнями. Кожен рівень містить один відсортований сегмент. Дані починаються з рівня 0, а потім об'єднуються в цикл рівня 1. Згодом сегменти рівня 1 об'єднується в сегмент рівня 2 і так далі. Кожен рівень обмежений у своїх розмірах. Фактор росту k вказується як збільшення розміру даних на кожному рівні. Ми можемо проаналізувати LSM-дерево на основі рівня наступним чином. Якщо фактор росту є k, а найменший рівень — це один файл розміром B, то кількість рівнів дорівнює $O(\log_k N/B)$, де N – розмір бази даних. Дані потрібно перемістити з кожного рівня один раз, але дані з даного рівня неодноразово об'єднуються з даними з попереднього рівня. У середньому після першого запису на рівень кожен елемент даних повертається назад на той самий рівень приблизно k/2 рази. Отже, загальне посилення запису становить $O(k * \log_k N/B)$. Щоб виконати запит короткого діапазону в холодному кеші, ми повинні виконати двійковий пошук на кожному з рівнів. Для найвищого рівня розмір даних становить O(N), тому читання виконує $O(\log\ N/B)$ читань з диску. На минулому рівні розмір даних дорівнює O(N/k), тому для читанння з нього виконується $O(\log\ N/(kB))$ читань з диску. Таким чином, можемо вирахувати загальну кількість читань з диску: $O((\log^2 \frac{N}{B})/\log k)$.

Таблиця 3.3.

Порівняння метрик продуктивності бінарного дерева та журнальованого індекса

№	Структура даних	Посилення запису	Посилення читання
1	Бінарне дерево	O(B)	$O(\log_B N/B)$
2	Журнальований індекс	$O(k * \log_k N/B)$	$O((\log^2 \frac{N}{B})/\log k)$

Окрім теоретичного аналізу, можливо отримати також практичні. Для проведення замірів продуктивності двох структур даних, що порівнюються, проведемо експеримент на базі даних, що дозволяє створювати таблиці що базуються як на бінарному дереві, так и на журнальованому індексі – WiredTiger. WiredTiger — це високопродуктивна, масштабована, транзакційна, високоякісна, NoSQL система управління базами даних. Наразі ця система управління базами даних є основним компонентом бази даних MongoDB.

Тест створює базу даних шляхом послідовної вставки 100 мільйонів пар ключ/значення з розміром значення 100 байт. Генерує близько 10 ГБ даних.Тест запускає один потік, який виконує вставки, і набір потоків, які одночасно виконуватимуть точкові запити. На наступних графіках кожна фаза робочого навантаження відображається протягом 10 хвилин.

Між виконаннями тестів змінюються такі умови:

- Чи обмежена кількість запитів за секунду
- Кількість потоків, які виконують запити

Результати проведених експериментів та замірів продуктивності двох методів зберігання та індексації даних зображено на Рис. 3.1-3.4.

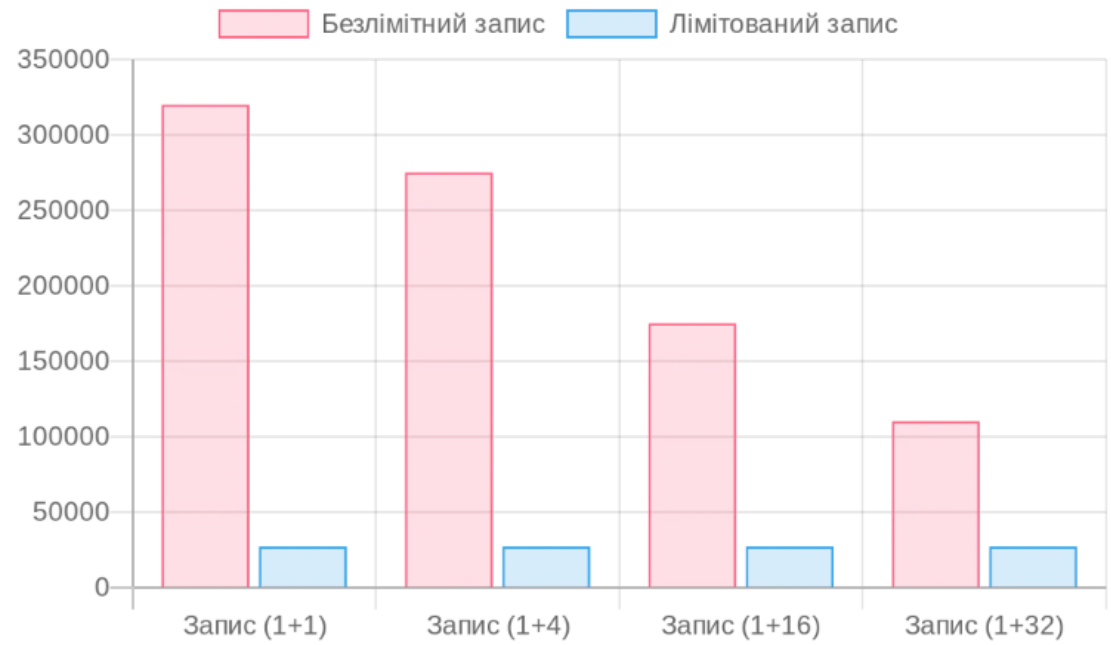

Рис. 3.1. Вимірювання продуктивності операцій запису в журнальований індекс

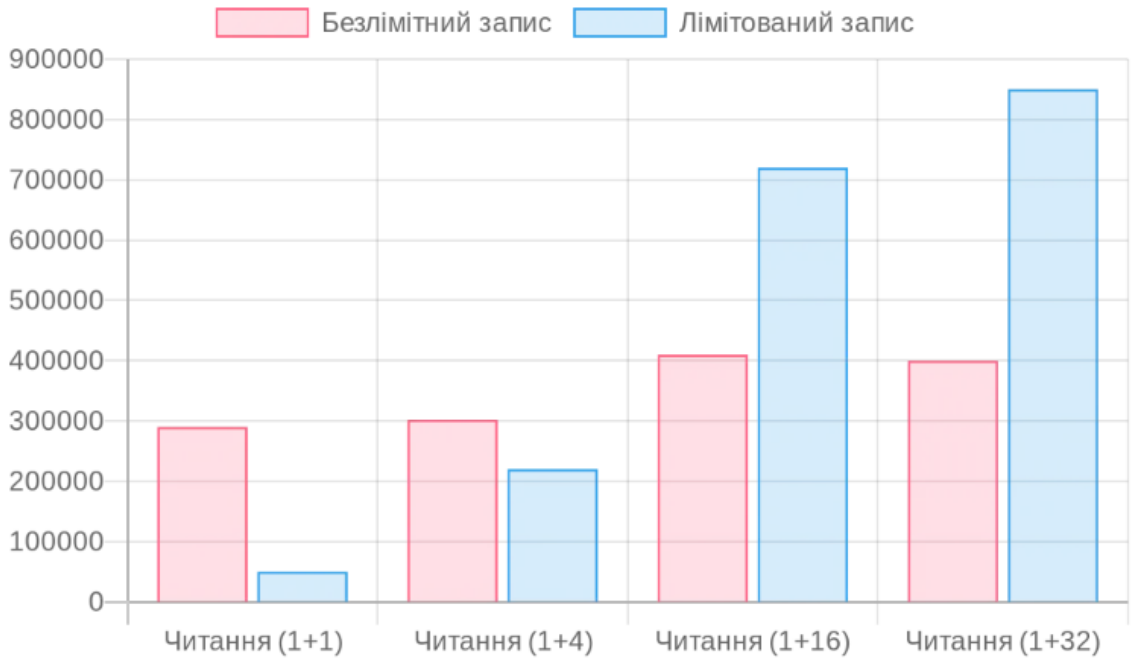

Рис. 3.2. Вимірювання продуктивності операцій читання з журнальованого індексу

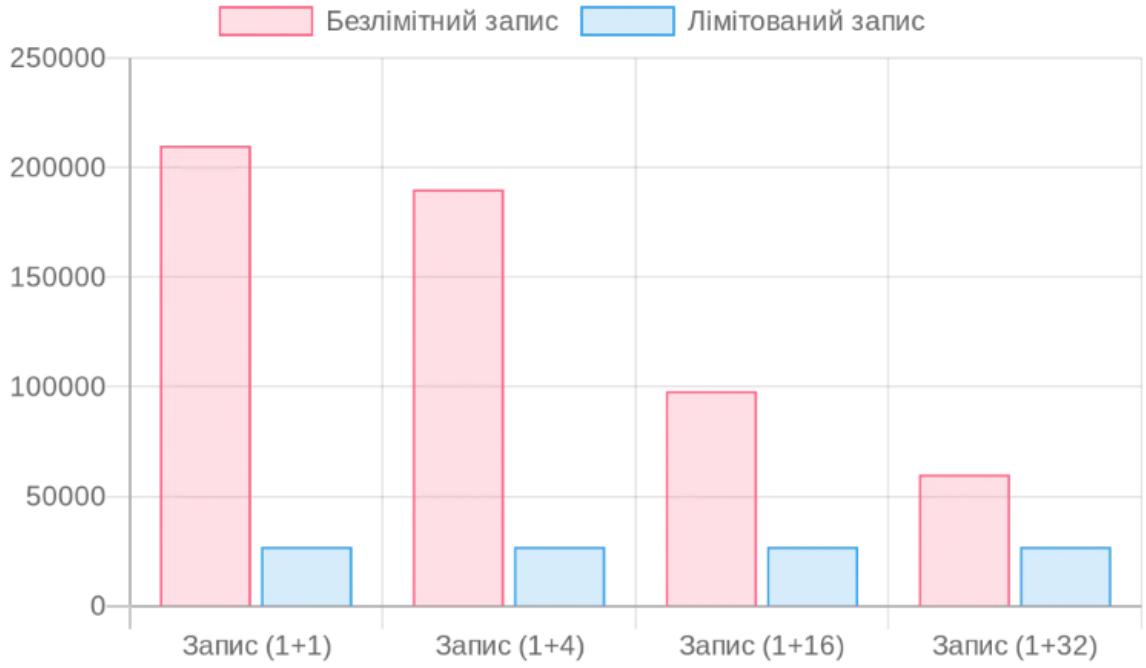

Рис. 3.3. Вимірювання продуктивності запису в бінарне дерево

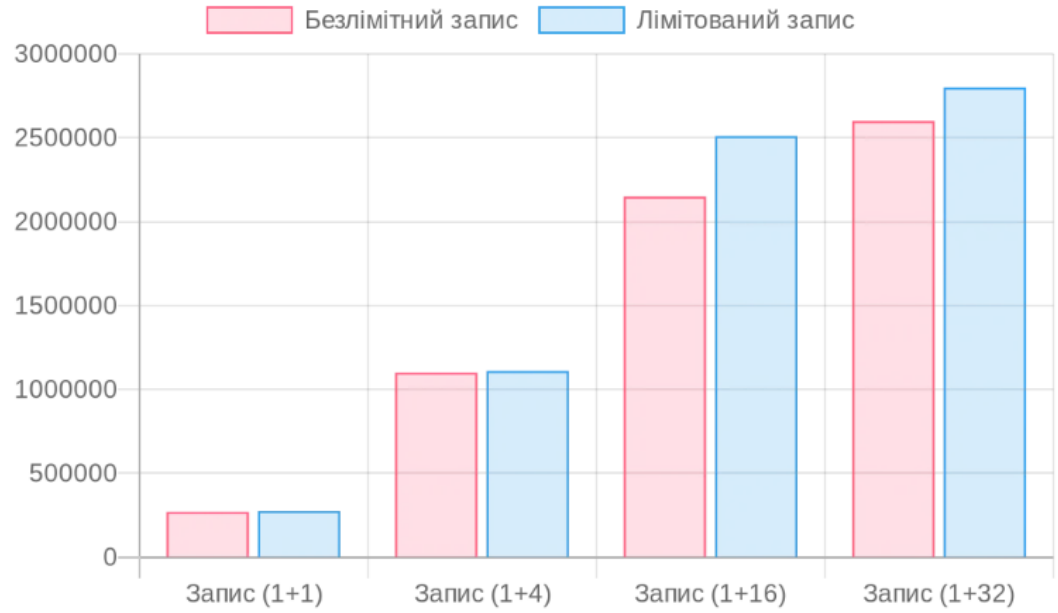

Рис. 3.4. Вимірювання продуктивності читання з бінарного дерева

Зауважимо, що обидві структури даних здатні підтримувати 30 000 вставок за секунду, які вимагаються обмеженим робочим навантаженням. Журнальованому індексу вдається перевищувати пропускну здатність вставки бінарного дерева в 1,5-2 рази. Це очікувано – журнальований індекс оптимізовано для великих

навантажень запису. Також цікаво, що існує лише один потік запису, але зі збільшенням кількості потоків, які виконують читання, пропускна здатність запису зменшується. Ефект досить вражаючий з 32 потоками читання — що й слід було очікувати, оскільки машина, на якій працює тест, має 16 ядер ЦП — наявність 32 потоків призводить до перемикання контексту.

Стосовно читання, ключове спостереження між двома графіками полягає в тому, що пропускна здатність читання за допомогою бінарного дерева в 1,5-3 рази більше, ніж за допомогою дерева журнальованого індекса. Різниця стає більш помітною, коли додається більше потоків читання. Перший набір результатів для журнальованого індексу є побічним ефектом того, як виконується тест. Тест запускає фазу заповнення, за якою відразу йде послідовність фаз читання. У журнальованому дереві фаза заповнення залишить дерево в стані, коли воно має багато фонових завдань злиття сегментів.

3.1.4. Вибір та аргументація

Бінарні дерева глибоко вкоренилися в архітектурі БД і забезпечують незмінно гарну продуктивність для багатьох видів навантаження, тому малоймовірно, що найближчим часом від них відмовляться. А в нових сховищах даних стають дедалі популярнішими журнальовані індекси. Не існує швидкого і простого правила, яке б дозволило вирішити, який тип підсистеми зберігання краще в конкретному випадку, це доводиться перевіряти досвідченим шляхом.

Проте, індекс на основі бінарного дерева повинен записувати кожен елемент даних принаймні двічі: один раз у журнал із попереджувальним записом і другий на саму сторінку дерева (і ймовірно, знову при розбитті сторінок). Крім того, виникають додаткові накладні витрати через те, що записувати доводиться відразу всю сторінку, навіть якщо в ній змінилося лише кілька байтів. А деякі підсистеми зберігання перезаписують ту саму сторінку двічі, щоб не залишитися з не повністю оновленою сторінкою у разі збою живлення [5].

Журнальовані індекси теж переписують дані кілька разів через багаторазове ущільнення та злиття сегментів. Це питання особливо важливе у випадку SSD, для

яких посилення запису є деструктивним адже вони здатні переписувати блоки лише обмежену кількість разів до повного зношування. Але для деяких систем, що вимагають великих обсягів запису даних, вузьким місцем по продуктивності здатна виявитися швидкість, з якою база записує дані на диск. У цьому випадку посилення запису безпосередньо впливає на продуктивність: чим більше підсистема зберігання записує на диск, тим менше операцій запису в секунду вона може виконати в рамках доступної їй смуги пропускання диска.

Але журнальовані індекси зазвичай здатні забезпечити більшу пропускну здатність, ніж бінарні дерева, частково через їх слабше посилення запису (хоча це залежить від налаштувань підсистеми зберігання та навантаження), а частково - через те, що вони послідовно записують компактні файли сегментів замість перезапису кількох сторінок дерева [6]. Описана різниця особливо суттєва для магнітних жорстких дисків, на яких послідовні операції запису працюють набагато швидше.

Проведені експерименти дозволили зрозуміти, що журнальований індекс дозволяє підвищити пропускну здатність для запису даних у 1.5-2 рази. Оскільки OLAP бази даних спрямовані на збір великих даних із метою подальшого аналізу, ми можемо зробити висновок, що журнальований індекс є більш доречним для використання у таких базах даних, адже кількість операцій запису у них кратно перевищує кількість аналітичних запитів на читання.

3.2. Методи зберігання даних. Стовпчикове та рядкове зберігання.

Таблиці в аналітичних базах даних часто відрізняються великою кількістю

```
SELECT
  dim_date.weekday, dim_product.category,
  SUM(fact_sales.quantity) AS quantity_sold
FROM fact_sales
  JOIN dim_date    ON fact_sales.date_key   = dim_date.date_key
  JOIN dim_product ON fact_sales.product_sk = dim_product.product_sk
WHERE
  dim_date.year = 2013 AND
  dim_product.category IN ('Fresh fruit', 'Candy')
GROUP BY
  dim_date.weekday, dim_product.category;
```

рядків, адже зберігають велику кількість метаданих операцій та різноманітних дій користувачів, тощо. Проте, хоча ширина таблиць часто перевищує 100 стовпців, типовий запит до такої таблиці використовує лише чотири чи п'ять із них за один раз. Візьмемо для приклада запит, що аналізує дані велокої кількості рядків таблиці, що містить дані о покупках по датам. Цей запит дозволяє дізнатися, які товари покупаються частіше у визначені дні тижня, Рис. 3.5:

Для виконання цього запиту, базі даних потрібно обробити кожну строку із таблиці покупок за 2013 рік. Чи зможемо ми виконати цей запит ефективно, якщо дані збережено у класичних атомарних сторінках розміром 4КБ, а для зчитування лише трьох колонок із таблиці fact_sales доведецься зситувати весь рядок таблиці,

Рис. 3.5. SQL код аналітичного запиту, що знаходить найпопулярніші товари в окремі дні тижня.

яка містить терабайт даних?

У більшості класичних транзакційних базах даних дані розміщуються рядково: всі значення з одного рядка таблиці зберігаються поряд один з одним. Документоорієнтовані БД влаштовані аналогічно: весь документ зазвичай зберігається у вигляді безперервної послідовності байтів.

Щоб виконати такий запит, як у прикладі вище, нам знадобляться індекси по стовпцях fact_sales.date_key та/або fact_sales.product_sk, які повідомляли б підсистемі зберігання, де шукати всі продажі для конкретної дати або конкретного

товару. Але навіть після цього реалізована рядково підсистема зберігання повинна завантажити всі рядки (кожний з яких складається з більш ніж 100 стовпців) з диску в оперативну пам'ять, виконати їх синтаксичний розбір і відфільтрувати ті, що не задовольняють заданим умовам. На це може піти багато часу. Ідея стовпчикових сховищ проста: потрібно зберігати поряд значення не з одного рядка, а з одного стовпця. Якщо кожен стовпець зберігається в окремому файлі, то запиту потрібно лише прочитати і виконати синтаксичний розбір необхідних запиту стовпців, що може заощадити масу зусиль. Ця ідея проілюстрована нижче. Примустимо, ми маємо такий вміст таблиці продажів:

Таблиця 3.4.

Таблиці бази даних, що містить дані про придбані товари.

№	date	product	promotion	customer	qty	net_price	discount	store
1	231113	69	NULL	NULL	1	13.99	13.99	4
2	231113	69	19	NULL	3	14.99	9.99	5
3	231113	69	NULL	191	1	14.99	14.99	5
4	231113	74	23	202	5	0.99	0.89	3
5	231114	31	NULL	NULL	1	2.49	2.49	2
6	231114	31	NULL	NULL	3	14.99	9.99	3
7	231114	31	21	123	1	49.99	39.99	3

Тоді файли рядків, що зберігають дані, виглядали б наступним чином:

- Файл рядку ‘date’: 231113, 231113, 231113, 231113, 231114. 231114, 231114.
- Файл рядку ‘product’: 69, 69, 69, 74, 31, 31, 31.
- Файл рядку ‘promotion’: NULL, 19, NULL, 23, NULL, NULL, 21.
- Файл рядку ‘customer’: NULL, NULL, 191, 202, NULL, NULL, 123.
- Файл рядку ‘qty’: 1, 3, 1, 5, 1, 3, 1.
- Файл рядку ‘net_price’: 13.99, 14.99, 14.99, 0.99, 2.49, 9.99, 39.99.
- Файл рядку ‘store’: 4, 5, 5, 3, 2, 3, 5.

Розміщення даних по стовпцях вимагає, щоб файли всіх стовпців містили рядки в однаковому порядку. Отже, при необхідності зібрати воєдино цілий рядок можна взяти зі всіх файлів окремих стовпців 23-й елемент і зібрати їх разом для формування 23-го рядка таблиці.

3.2.1. Стискання стовпців

Окрім завантаження з диску лише тих столбців, які потрібні для запиту, можна ще більше знизити вимоги до пропускної здатності диска, сжав дані. Нащастя, стовпчикове сховище часто дуже добре піддається стисканню. Розглянемо послідовність значень для стовпчика ‘product’ з минулого пыдрозділу: схоже, що вони часто повторюються, і це хороший знак у сенсі можливості стискання. В залежності від даних, що містяться в таблиці, застосовуються різні методи стискання. Один із методів, особливо ефективних при роботі зі складами даних, — кодування за допомогою бітової мапи (bitmap encoding):

Рис. 3.6. Приклад кодування даних стовпця “product” із використанням бітової мапи

На рисунку вище наведений приклад кодування можливих значень стовпця задопомоги бітової маски. Бітова маска створюється під кожне з можливих значень, та має значення 1 для рядків, що містять значення, яке закодоване цією маскою. Тож маємо 6 бітових масок, по одній для кожного унікального значення стовпця. Завдяки бітовим маскам, замість зберігання 18 цілих чисел, мі можемо обійтися зберіганням лише 6 + 1 біт на кожну строку у таблиці, при цьому самі біти маски

можуть бути закодовані послідовностями, наприклад дані вищенаведеного рядку можуть бути закодовані наступним чином:

- Product = 29: 9, 1 (9 нулів, 1 одиниця, інше - нулі).
- Product = 30: 10, 2 (10 нулів, 2 одиниці, інше - нулі).
- Product = 31: 5, 4, 3, 3 (5 нулів, 4 одиниуі, 3 нулі, 3 одиниці, інше - нулі).
- Product = 68: 15, 1 (15 нулів, 1 одиниця, інше - нулі).
- Product = 69: 0, 4, 12, 2 (0 нулів, 4 одиниці, 12 нулів, 2 одиниці).
- Product = 74: 4, 1 (4 нулі, 1 одиниця, інше - нулі).

Зазвичай у стовпці кількість різних значень є невеликою в порівнянні з кількістю рядків (наприклад, у розничного торговця може бути мільярди торгових операцій, але тільки 10 000 різних товарів). Для ефективного представлення таких стовпців ми можемо перетворити їх у набір бітових карт: окрему карту для кожного унікального значення, де кожен біт відповідає рядку. Якщо значення міститься в рядку, біт встановлюється в 1, в іншому випадку - залишається 0. При невеликому значенні n (наприклад, у стовпці "Країни" може бути приблизно 200 різних значень), такі бітові карти можна ефективно зберігати в одному біті на рядок. Проте, якщо n значно більше, то більшість бітових карт буде містити безліч нулів (вони називаються розрізненими). У цьому випадку бітові карти можна додатково стиснути за допомогою алгоритму кодування довжин серій, як показано вище.

Значущим обмеженням для запитів до сховищ даних, які повинні переглядати мільйони рядків, є пропускна здатність переміщення даних з диска в пам'ять. Однак це не єдине таке обмеження. Крім того, розробникам аналітичних баз даних доводиться турбуватися про ефективне використання швидкості передачі даних з оперативної пам'яті в кеш процесора, уникання помилкового передбачення гілок та "пузирів" в конвеєрі обробки інструкцій CPU, а також використання векторних інструкцій (single-instruction-multi-data, SIMD) сучасних процесорів [7, 8].

Крім зменшення обсягів завантажених з диска даних, схеми зберігання стовпцевого типу також можуть служити для більш ефективного використання циклів CPU. Наприклад, підсистема обробки запитів може взяти порцію даних, яка

ідеально поміщається в L1-кеш процесора, і обробляти її в неперервному циклі (тобто без викликів функцій). Процесор здатний виконувати такий цикл значно швидше, ніж код, що містить багато викликів функцій та умов для кожного оброблюваного запису. Стовпчикове стиснення дозволяє помістити в тому ж обсязі L1-кеша більше рядків для одного стовпця. Для роботи безпосередньо з такими порціями стиснених стовпцевих даних можна використовувати такі оператори, як описані вище побітові І та АБО. Цей метод відомий як векторизована обробка [9].

3.2.2. Продуктивність стопвчикового сховища

Стовпчикове зберігання дозволяє ігнорувати всі дані, які не відносяться до конкретного запиту, оскільки можна отримати інформацію лише з тих стовпців, які потрібні. У порівнянні з рядково-орієнтованою базою даних, якщо, наприклад, запит працює з рядками і повинен знайти середню щільність населення в містах з населенням понад мільйон людей, запит буде отримувати доступ до кожного запису в базі даних (тобто всіх її полів), щоб отримати інформацію з двох стовпцівю. Такий алгоритм зменшує кількість даних, що потрібно зчитати з диску для виконання запиту.

Для порівняння проудктивності такого способу здебережжня даних, було написано програму, що робить вставку в дві бази даних. Для експерименту було обрано:

- PostgreSQL – безкоштовна база даних з відкритим вихідним кодом, яка є динамічною та розширюваною. Використання варіюється від невеликих програм до сховищ даних. Ця база зберігає дані у класичному
- ClickHouse – розглянуте у першому розділі відкрите програмне забезпеченням для управління колоночними базами даних, яке спеціалізується на здійсненні швидких аналітичних запитів на великих обсягах даних.

Для проведення порівняльного тесту було використано базу даних із приблизно 135 мільйонами записів журналів веб-подій, розподілених у чотирьох різних таблицях.

Бази даних використовувалися локально, на локальній машині з процесором i5-12500H із тактовою частотою 2.5 ГГц у поєднанні з 32 ГБ оперативної пам’яті.

Було створено набір запитів, які зазвичай вважаються аналітичними (тобто зосереджені на стовпцях, а не на рядках). Щоб зробити процес максимально однорідним, для кожного виконаного запиту було очищено кеш відповідного механізму зберігання та перезапускали машину між змінами баз даних.

Для рядкових баз даних було створено найефективніші індекси для тестових запитів. Для стовпчикового сховища не було створено індексів, окрім того, який був створений за замовчуванням. Результати відображені в таблиці нижче.

Таблиця 3.5.

Швидкість виконання запитів к PostgreSQL та ClickHouse

№	Запит	PostgreSQL	ClickHouse
1	Q1: Загальна кількість подій	92	< 1
2	Q2: Кількість подій типу A	94	< 1
3	Q3: Розподіл кількості подій по днях	87	3
4	Q4: Розподіл кількості подій по операції	96	< 1
5	Q5: Топ 10 подій по операції	91	< 1
6	Q6: Кількість подій що вміщають зображення	99	4
7	Q7: Кількість подій що вміщають скрипти	276	14

Розглядаючи наведені вище дані, не дивно, що стовпчикова база даних має значно коротший час виконання порівняно з OLTP базою даних. Хоча ClickHouse

не підтримує об'єднання таблиць в основному, цю проблему можна обійти, використовуючи підзапити. На наведеному нижче зображенні представлено порівняльний погляд на ці бази даних та їх продуктивність з аналітичними запитами (менше значення - краще):

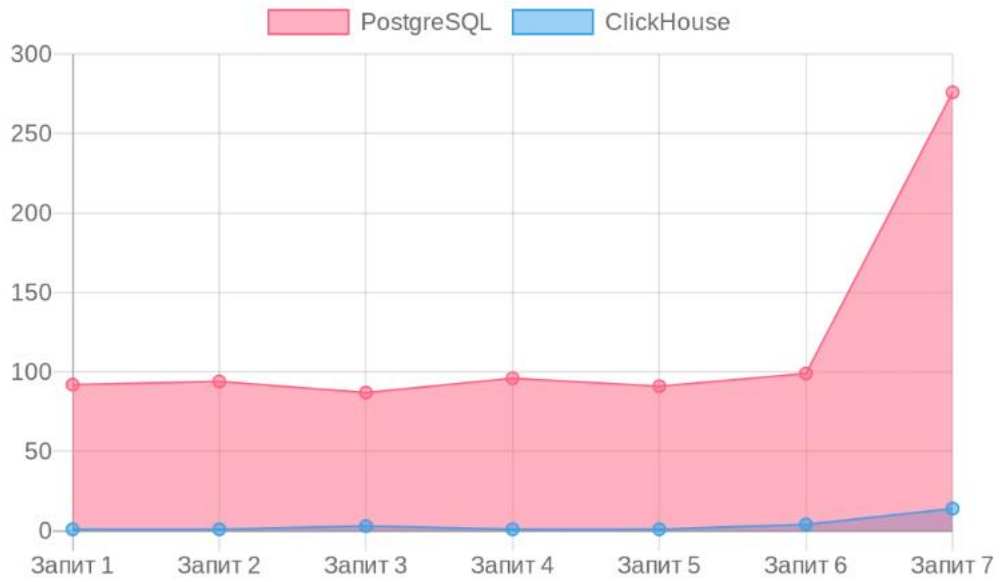

Рис. 3.7. Порівняння швидкості виконання запитів у PostgreSQL та ClickHouse

3.2.3. Ефективність збереження даних

Як можна побачити у таблиці порівнянь, колоночне зберігання має вдосконалені механізми стиснення порівняно з рядковими базами даних. Це відбувається через те, що кожен стовпець, стиснутий як окремий файл, має унікальний тип даних. Крім того, у таких баз даних немає індексів, крім того, що створюється за замовчуванням для первинного ключа.

Такі особливості дозволяють використовувати простір для зберігання дуже ефективно, як показано на графіку нижче: ClickHouse займає 10 ГБ простору, за ним йдуть сирі дані - 76 ГБ, і, нарешті, PostgreSQL займає 78 ГБ простору зберігання. Це на 780% більше, ніж у ClickHouse (рис.3.8.).

Варто зауважити, що PostgreSQL ззаймає навіть більше місця, аніж сирі дані. Це траплається тому, що нами було додано вторинні індекси для ефективності виконання запитів, а кожен індекс за своєю суттю є копією деяких даних із посиланнями на місцезнаходження в сховищі за первинним індексом.

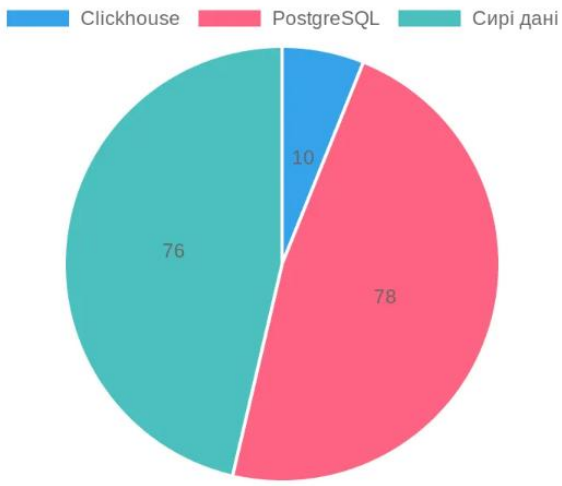

Рис. 3.8. Порівняння ступеня стискання даних у PostgreSQL та ClickHouse

3.2.4. Аргументація вибору

Стовпчиковий метод зберігання вражає своїми можливостями: обробка величезних обсягів даних за кілька секунд або навіть миттєво. Існує безліч прикладів (ClickHouse, Redshift, BigQuery, Snowflake, MonetDB, Greenplum, MemSQ та інші), які мають оптимальну продуктивність для аналітики.

Але така базова архітектура також вносить помітний недолік: завантаження даних. Такий сбосіб зберігання показує слабку продуктивність для змішаних робочих завдань, які вимагають обробки даних у реальному часі та високої пропускної здатності. Поєднання швидкості вставки та запиту – це найважливіша задача OLAP систем управління базами даних. Ще одним недоліком стовпчикового сховища є неможливість легкого блокування сегменту даних для виконання транзакції: у випадку баз даних на основі бінарних дерев, під час транзакції блокування відбувається на рівні блоків, що містять рядки цілком; у випадку ж стовпчикових баз даних, дані кожного окремого стовпчика зберігаються в окремому файлі, тож механізм блокування є набагато складнішим та має свої обмеження.

Проте, у поєднанні з журнальованим індексом, розглянутим у розділі 3.1, база даних має можливість знайти баланс між вставкою та читанням, та

забезпечити можливість зберігання великих даних, що додаються до бази дуже часто та у великих обсягах, і при тому дозволити аналізувати ці дані у реальному часі.

3.3. Реплікація та секціонування даних

На перший погляд реплікація асоціюється із копіюванням даних, а секціонування — з розподілом. Проте у контексті аналітичних баз даних ці поняття означають більш глибокі стратегії роботи з великими обсягами інформації. Відокремлення та розміщення даних стає критичним для виконання складних аналітичних операцій та швидкості їх виконання.

Роль реплікації у забезпеченні доступності та відмовостійкості: Реплікація — це стратегія, яка забезпечує наявність даних на різних серверах, зменшуючи ймовірність втрати інформації при відмові одного з них. Це є необхідним для підтримки стабільної роботи системи при можливих випадках відмови серверів або збоїв. Секціонування, або розподілення, дозволяє ефективно розміщувати дані в різних локаціях для підвищення продуктивності та швидкості доступу. Це ключовий аспект в аналітичних базах даних, оскільки дозволяє оптимізувати доступ до даних під конкретні завдання та операції.

Незважаючи на переваги, реплікація та секціонування також викликають виклики, такі як синхронізація даних та збереження консистентності. Варто розглянути способи оптимізації процесів, а також шляхи подолання можливих недоліків цих підходів.

3.3.1. Реплікація даних

Реплікація передбачає зберігання копій тих самих даних на декількох машинах, які з'єднані мережею. Реплікація має велике значення для аналітичних баз даних з кількох ключових причин, які впливають на ефективність та надійність аналітичної системи:

- Забезпечення доступності та надійності. Реплікація дозволяє зберігати кілька копій даних на різних серверах чи вузлах. Це значно підвищує надійність системи: якщо один сервер вийде з ладу, інші репліки

продовжать надавати обслуговування. У контексті аналітичних баз даних, де швидкий доступ до великого обсягу інформації є важливим, реплікація забезпечує неперервну доступність даних для аналітичних операцій та звітності.

- Зменшення часу відновлення. В разі відмови обладнання або випадку непередбаченої ситуації, реплікація дозволяє відновити доступ до даних швидко та безпечно. Наявність кількох копій даних дозволяє швидко відновити роботу системи без значного втрати інформації або простою у роботі.
- Покращення продуктивності. Реплікація може допомогти покращити продуктивність системи. Наприклад, за допомогою розподілення навантаження на різні репліки, можна підвищити швидкість відповіді на запити та розподілити навантаження на обробку даних між різними серверами чи вузлами.
- Зберігання даних віддалено. Реплікація дозволяє зберігати копії даних віддалено, навіть у різних географічних регіонах. Це корисно для забезпечення стійкості до регіональних проблем, таких як природні катастрофи або збої мережі, та для забезпечення даних для використання в децентралізованих організаціях.
- Використання реплікації у реальному часі. В деяких випадках, де важливо отримувати дані в реальному часі, реплікація дозволяє надавати актуальні дані на декількох рівнях системи одночасно, що полегшує аналітику в режимі реального часу.
- Оптимізація навантаження. Реплікація може використовуватися для розподілення навантаження на читання та запис, поліпшуючи швидкодію системи. Така оптимізація може бути важливою для систем, що обробляють великі обсяги інформації.

При реалізації реплікації даних, потрібно звернути увагу на низку принципів, на яких вона будується:

- Дублювання даних. Копіювання одних і тих самих даних на кількох вузлах або серверах для забезпечення доступності та надійності.
- Автономність. Кожна репліка незалежна, тобто може працювати самостійно, що забезпечує безперервну роботу системи навіть при відмові одного сервера.
- Синхронізація. Підтримання однакових даних на всіх репліках в реальному часі або з невеликою затримкою для забезпечення консистентності.

3.3.2. Секціонування даних

У випадку великих обсягів даних або обробки значних обсягів даних реплікації недостатньо: необхідно розбити дані на секції, іншими словами, виконати шардування даних.

Зазвичай секції визначаються так, що кожен елемент даних (запис, рядок або документ) належить рівно до однієї секції. Цього можна досягти за допомогою різних методів. Фактично кожна секція сама по собі є маленькою базою даних, хоча база може підтримувати операції, які охоплюють одразу декілька секцій.

Основна мета секціонування даних - масштабованість. Різні секції можна розмістити на різних вузлах у кластері, що не передбачає розділення ресурсів. Отже, великий набір даних можна розподілити по багатьом жорстким дискам, а запити - по багатьом процесорам. При запитах до однієї секції кожен вузол здатен незалежно виконувати запити в своїй секції, таким чином пропускну здатність по запитах можна масштабувати просто додаванням нових вузлів.

Великі, складні запити можна паралельно обробляти декількома вузлами, хоча це набагато складніше. Секціоновані бази даних з'явилися в 1980-х роках. Це були такі програмні продукти, як Teradata і Tandem NonStop SQL[10]. Недавно технологія була відкрита знову базами даних NoSQL. Деякі з систем спроектовані з огляду на навантаження у вигляді обробки транзакцій, а інші призначені для аналітики. Це впливає на налаштування системи, але основи секціонування не відрізняються для обох видів навантаження.

При реалізації секціонування даних, потрібно звернути увагу на низку принципів, на яких воно будується:

- Розділення даних. Розподіл великої бази даних на менші частини (секції) для покращення швидкодії та масштабованості.
- Горизонтальний та вертикальний розподіл. Горизонтальний - розділення на основі рядків; вертикальний - розділення на основі стовпців.
- Рівномірність. Доцільне розподілання навантаження між секціями.

3.4. Висновки

Алгоритми та структури даних є найважливішим підгрунтям до розроби програмного забезпечення – будь-яка програма розробляється, спираючись на алгоритми того чи іншого рівня в ієархії програмного забезпечення, починаючи від алгоритмів що ховаються за компілятором язика, на якому програма розробляється, закінчуючи функціями операційної системи, в якій програма виконується.

Після першого та другого розділів цієї квалійакайійної роботи, в яких було розглянуто сучасні рішення аналітичних баз даних, та визначено вимоги до таких баз, включно із створенням ТЗ, цей розділ досліджує алгоритми, що покривають ці вимоги, порівнює їх між собою, аргументує вибір алгоритмів, та надає методичні вказівки з їх реалізації на високому рівні. Робота, проведена в цьому розділі, дозволяє зрозуміти різницю між транзакційними та аналітичними базами даних у порівнянні вже не вимог, а методів їх задоволення.

Завдяки дослідженню бінарних дерев та журнальованих індексів, стає очевидним що саме журнальовані індекси є найкращим кандидатом на використання у аналітичних базах даних, в той час як бінарні дерева були розповсюджені у реалізаціях первинних індексів через вимоги до транзакційності та високої консистентності даних. Заміри продуктивності алгоритмів дозволяють зрозуміти слабкі та сильні сторони журнальованих дерев, які можна використати у аналітичних базах даних, та які вже використовуються в більшості сучасних з них.

З огляду на великі обсяги даних, велику кількість збережених покупок, товарів, переміщень, дій та телеметрії, дослідження стовпчикового методу зберігання даних дозволяє оцінити його внесок в можливість реалізації сховища, що стискає дані набагато єфективніше ніж більшість транзакційних баз даних, що розроблялися для значно менших обсягів інформації. Завдяки стовпчиковому зберіганню, дані можуть стискатися на майже 800% ефективніше, ніж при використанні класичного зберігання по сторінкам. Окрім того, цей метод дозволяє виконувати аналітичні запити у сотні раз швидше завдяки тому, що база даних читає з диску лише ті дані, що потрібні для виконання запиту (аналітичні запити найчастіше агрегують лише 4-5 стовпців),а також завдяки векторизованій обробці зі сторони CPU.

Також важливим додатковим механізмом сучасних аналітичних баз даних є механізм реплакіції та секціонування даних, який дозволяє розподіляти навантаження між багатьма серверами, тобто масштабувати базу даних горизонтально, а це, в свою чергу, дозволяє виконувати запити та реагувати на зміни в бізнес-моделі компанії набагато швидше.

РОЗДІЛ 4
ПРОЕКТУВАННЯ АРХІТЕКТУРИ. ПРОТОТИП СИСТЕМИ

Після аналізу вимог до системи, детального розгляду алгоритмів та механізмів, які забезпечують їх задоволення, настав час перейти до ключового етапу — проектування архітектури та створення прототипу системи. Глибокий розгляд журнальованих дерев злиття як первинного індексу, стовпчикового зберігання даних, а також реплікації та секціонування визначив важливі основи для подальшої розробки системи.

Ключові аспекти, які будуть висвітлені в цьому розділі:

- Проектування архітектурних моделей. Розробка структури системи на основі аналізу вимог та вибраних алгоритмів. Вибір оптимальних типів архітектур та шаблонів проектування, які відповідають функціональним та нефункціональним вимогам.
- Створення прототипу системи. Розроблення початкової версії системи, що відображає ключові функції та можливості. Прототип допоможе перевірити правильність обраних концепцій та зрозуміти їх переваги та недоліки.
- Важливі аспекти механізмів. Детальний розгляд реплікації та секціонування даних, їхні особливості, методичні вказівки для ефективної реалізації та алгоритми, що використовуються для цих процесів.

Цей розділ відіграє ключову роль у створенні функціональної та оптимальної системи, що відповідає вимогам та має ефективну архітектуру.

4.1. Визначення компонентів системи

Визначення компонентів системи та побудова діаграми компонентів на етапі проектування OLAP бази даних має вирішальне значення для розуміння структури системи та організації роботи всіх її частин. Діаграма компонентів дозволить візуалізувати різні функціональні елементи системи та їх взаємодію. Це зробить

зрозумілішими ролі кожного компоненту та їхню взаємодію під час операцій, а також сприяє врахуванню та вирішенню потенційних проблем, що виникають під час роботи системи:

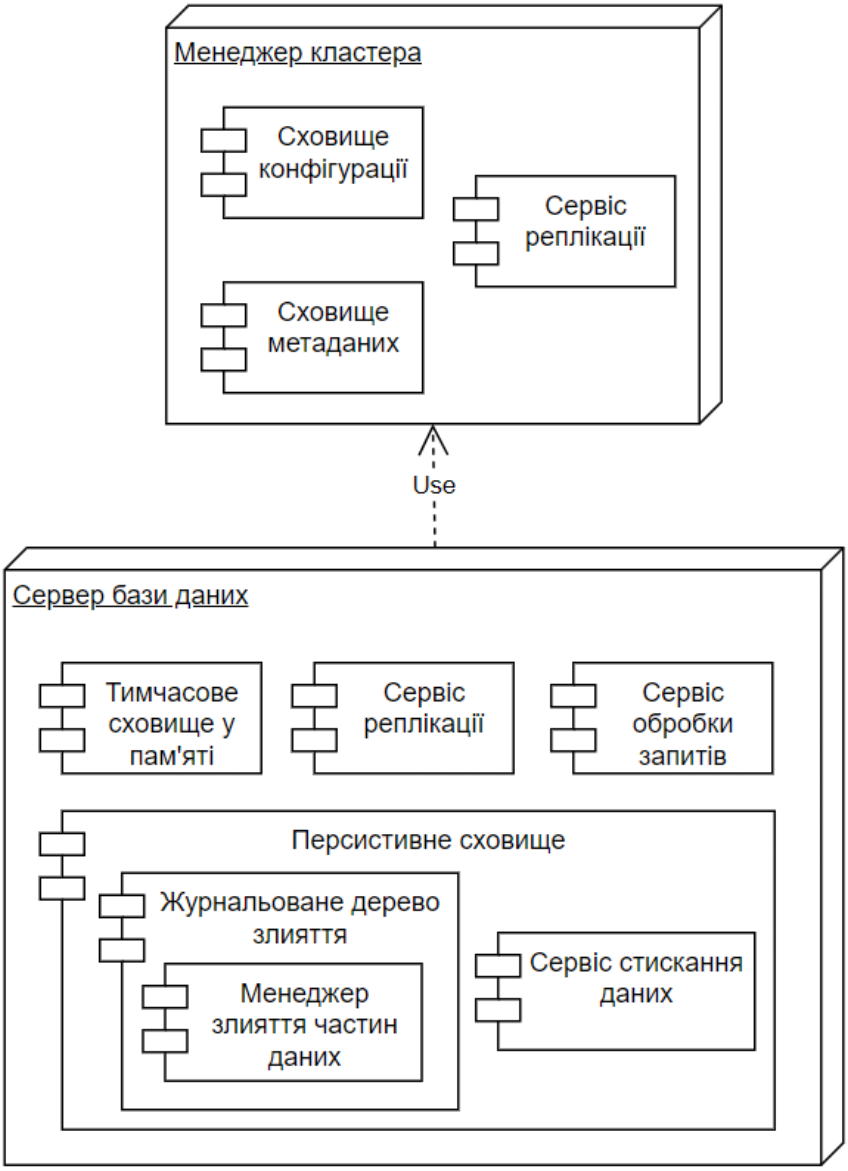

Рис. 4.1. Діаграма компонентів OLAP бази даних

Найвигіднішим з точки зору майбутнього розвитку системи буде виокремлення двох основних компонентів: Менеджера кластера та Сервера бази даних.

4.1.1. Менеджер кластера

Менеджер кластера розподіленої бази даних - це окреме програмне забезпечення, яке відповідає за управління розподіленою системою баз даних та координацію дій між різними частинами цієї системи. Основна мета менеджера кластера - забезпечити надійну та ефективну роботу бази даних у режимі розподіленого доступу, де дані можуть знаходитися на різних серверах.

Основними обов'язками менеджера кластера розподіленої бази даних є:

- Керування розподіленими вузлами. Менеджер кластера здійснює контроль за станом та доступністю різних вузлів бази даних. Він відповідає за збереження топології кластера, реагує на відмови вузлів та розподіляє навантаження між ними.
- Управління конфігураціями. Менеджер кластера відповідає за управління конфігураціями різних компонентів бази даних у кластері. Це означає збереження параметрів налаштувань, їх оновлення та узгодження між вузлами.
- Координація дій. Забезпечує синхронізацію дій між вузлами для уникнення конфліктів під час одночасної модифікації даних. Це важливо для забезпечення цілісності та унікальності даних в режимі розподіленого доступу.
- Обробка відмов та відновлення. Управління ситуаціями відмов та відновлення системи після них. Менеджер кластера забезпечує роботу системи в умовах відмов, виконує заміну вузлів, які вийшли з ладу, і відновлює доступність даних.

В системи, що проектується, Менеджер кластера планується використовувати для синхронізації стану кластера та реплікації даних, в той час як розподіл навантаження буде проводитися самим сервером бази даних.

Зв'язок між Серверами бази даних та Менеджером відбуватиметься через мережевий протокол, який базується на TCP/IP. Задля полегшення прототипізації, на першій стадії використовуватиметься HTTP протокол, який надаватиме змогу

Серверам бази даних отримувати потрібну інформацію від Менеджера за спроектованим інтерфейсом, що надається як API. Типовими операціями будуть:

- Запит конфігурації кластера
- Запит наявних частин таблиці (частин журнальованого дерева злиття), що синхронизовані між репліками у визначеній секції
- Запит списку реплік у визначеній секції, які мають потрібну частину таблиці (частину журнальованого дерева злиття), яка відсутня на цьому сервері, із ціллю її завантаження для синхронизації сховища

Таким чином, Менеджер виконуватиме функцію зв'язку між Серверами бази даних, що є обов'язковою вимогою для імплементації одного із визначених у главі 3 механізмів: репоікації та секціонування.

При реалізації Менеджеру кластера, рекомендується використовувати практики, застосовані у відомих менеджерах кластерів, таких як:

- Apache ZooKeeper. Використовується для управління та координації розподіленої системи. Він забезпечує засоби для синхронізації, конфігурації та надійності вузлів кластера. Цей менеджер використовується для кластерів ClickHouse, розглянутого у главах 1 та 3, хоча у новіших версіях був ореалізовано ClickHouse Keeper – самостійну систему.
- Etcd. Розподілене сховище ключей та значень, яке використовується для зберігання конфігурацій, даних та координації вузлів у кластері.
- Consul. Ще один інструмент для управління конфігураціями та координації роботи розподілених систем.

Ці менеджери кластера розподіленої бази даних використовуються для забезпечення надійності, швидкості та цілісності даних у розподіленому середовищі, де дані розміщені на різних вузлах.

4.1.2. Сервер бази даних

Сервер бази даних - це програмне забезпечення, яке відповідає за зберігання, управління та обробку даних. Він складається з різних модулів для забезпечення ефективності, надійності та швидкості роботи системи. Ось деякі з цих модулів:

- Тимчасове сховище у пам'яті. Цей модуль використовується для зберігання даних перед тим, як вони потрапляють до персистивного сховища. Тимчасове сховище розробляється із використанням червоно-чорного дерева, або інших видів дерев із можливістю балансування. Тимчасове сховище має лімит розміру (наприклад 4 мегабайти), після перевищення якого, тимчасове сховище записується у вигляді нової частини даних у окремий файл на диск, такий запис робиться послідовно за рахунок того, що дані в дереві вже є відсортованими. При запитах на читання, спочатку пошук відбувається у тимчасовому сховищі, а потім – у частинах даних у персистивному сховищі.
- Сервіс реплікації в аналітичних базах даних відіграє ключову роль у забезпеченні надійності та доступності даних. Реплікація означає створення та управління копіями даних на різних вузлах для забезпечення резервного копіювання та відновлення інформації в разі втрати або відмови основного сервера. Менеджер реплікації, такий як Zookeeper, використовується для координації та керування цим процесом. Він дозволяє системі створювати та підтримувати кластер з різними репліками даних, забезпечуючи їх синхронізацію та консистентність.
- Сервіс обробки запитів: Цей модуль відповідає за прийняття та обробку запитів користувачів або програм до бази даних. Він виконує різні операції, такі як вибірка, вставка, оновлення та видалення даних згідно з запитами користувачів.
- Персистивне сховище: Цей модуль відповідає за постійне зберігання даних на диску або іншому постійному носії. Дані зберігаються

частинами, кожна частина являє собою підмножину даних, відсортовану за первинним ключем, та збережену в окремому файлі.

- Журнальоване дерево злиття. Це структура даних, яка використовується для забезпечення консистентності даних із підвищеною швидкістю запису на диск завдяки тому, що вставка робиться великими блоками у послідовному порядку.
  - Менеджер злиття частин даних: Цей модуль відповідає за об'єднання та оптимізацію даних, що зберігаються у форматі журнальованого дерева злиття. Він є доволі простим та використовує алгоритм злияття двох відсортованих частин бази даних (так само як це робиться при сортуванні злияттям).

- Сервіс стискання даних: Використовується для стискання даних, що зберігаються у стовпчиковому сховищі. Це дозволяє зменшити обсяг зайнятого простору на диску та покращити продуктивність обробки даних.

4.2. Засоби та методи реалізації системи

Для успішної реалізації системи, визначимо ціль, з якою вона розробляється: створення прототипу, що дозволить наочно розглянути методологію розробки OLAP баз даних, та у майбутньому провести експерименти із алгоритмами, що використовуються.

Тож, для реалізації прототипу доцільно обрати просту та багатофункціональну мову програмування, багато базових механізмів, реалізацій алгоритмів та взаємодії із операційною системою для якої вже розроблено та представлено готовими бібліотеками. Такою мовою та платформою програмування в цьому проекті виступлять C# та .NET Core відповідно.

.NET є доцільним вибором для розробки прототипу OLAP бази даних з кількох причин.

По-перше, .NET має ряд переваг, серед яких - висока продуктивність, широкі можливості для розробки, швидкість та надійність. Він має велику кількість готових бібліотек та фреймворків, які значно спростять процес розробки прототипу, забезпечуючи доступ до широкого спектру інструментів для реалізації складних алгоритмів та взаємодії з операційною системою. Крім того, мова C# відома своєю простотою, читабельністю та широким спектром функціональних можливостей, що сприяє швидкій і ефективній розробці.

Компоненти системи “Менеджер кластеру” та “Сервер бази даних” повинні бути розділені як два окремих сервіси, що можуть працювати як різні процеси та на різних комп'ютерах.

Як було розглянуто у розділі 4, система передбачає такі основні алгоритми, які відрізняють та на яких базуються сучасні OLAP бази даних: індексація на базі журнальованого дерева, стовпчикове зберігання даних, стискання даних, реплікація та секціонування.

4.2.1. Індексація на базі журнальованих дерев

Індексація на базі журнальованого дерева передбачає використання такої структури даних як дерево, що балансується. В цьому проекті буде використано структуру даних червого-чорне дерево, що дозволить зберігати нові дані в оперативній пам'яті у відсортованому за первинним ключем вигляді.

Окрім цього, цей алгоритм має на увазі поступове збереження даних із оперативної пам'яті на диск у окремий файл, та поступове злиття таких окремих файлів у все більші та більші файли, що оптимізує використання дискового простіру. Алгоритм індексації на базі жернальованого дерева можна зобразити наступним чином:

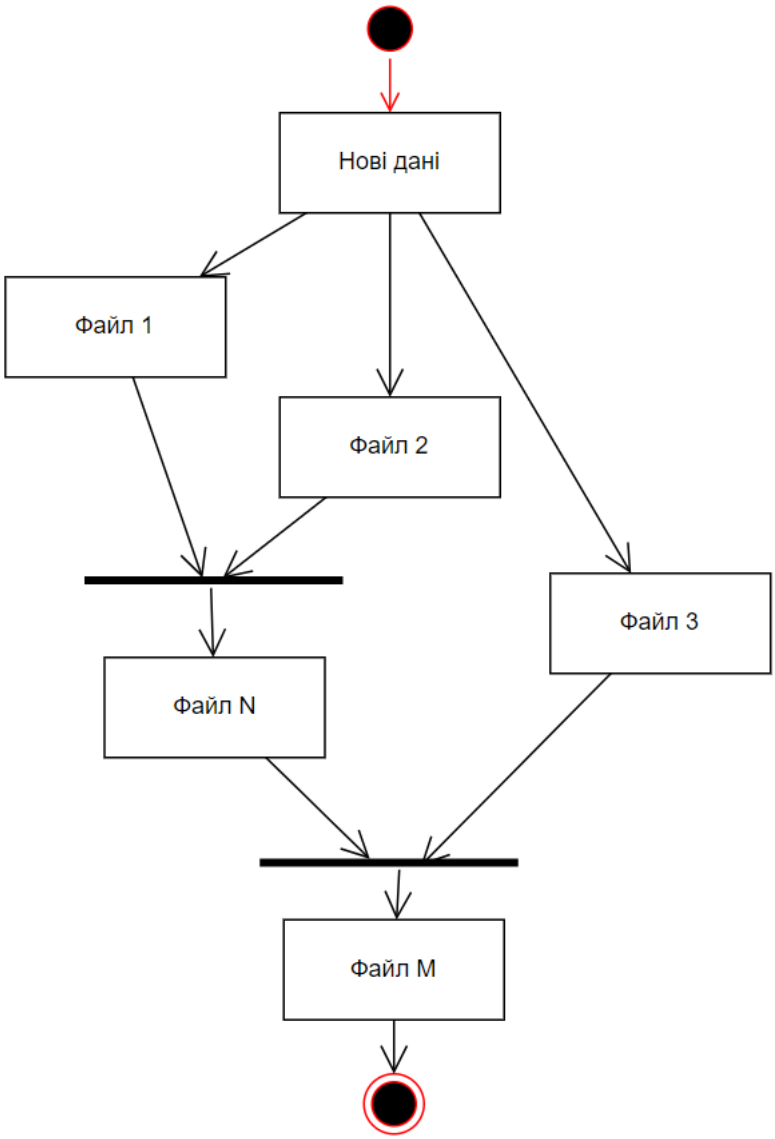

Рис. 4.2. Схема злиття даних з плином часу

4.2.2. Стовпчикове зберігання даних та стиснення

Одним з ключових алгоритмів, що використовуються у стовпчиковому зберіганні даних, є алгоритм стиснення. Зазвичай, дані у стовпчиках можуть мати велику кількість повторюваних значень, і стиснення допомагає зменшити обсяг пам'яті, не втрачаючи при цьому значимої інформації. Деякі алгоритми стиснення, такі як Run-Length Encoding (RLE), Dictionary Encoding та Delta Encoding, ефективно застосовуються для стовпчикових даних, зменшуючи обсяг пам'яті, який вони займають.

Крім того, для оптимізації операцій читання та фільтрації використовують структури даних, такі як Bitmap Indexes. Вони дозволяють швидко виконувати операції пошуку і вибору даних шляхом збереження бітових значень для кожного рядка, що вказує на наявність або відсутність певного значення в конкретному стовпці. Таким чином, стовпчикове зберігання даних в OLAP базах даних базується на ефективних алгоритмах стиснення та використанні оптимізованих структур даних для прискорення операцій пошуку та фільтрації.

Для реалізації рекомендоване використання методології, розглянутої у підрозділі 3.2.

4.2.3. Реплікація та секціонування даних

Для реалізації цих методів у системах OLAP використовують спеціалізовані алгоритми, що дозволяють ефективно керувати реплікацією та секціонуванням даних, забезпечуючи швидку та надійну роботу бази даних при великих обсягах інформації та навантаженнях.

Архітектура системи передбачає виокремлення логіки, що відповідає за синхронізацію серверів бази даних між собою, у окремий сервіс – Менеджер кластеру.

Реплікацію рекомендовано реалізувати за схемою “реплікації з одним ведучим вузлом” (“leader-based replication”), також відомою під назвою “реплікація

типа активний/пасивний” (“active/passive replication”). Ця схема вигладє наступним чином:

- Одна з реплік призначається ведучим вузлом. Клієнти, які бажають записати дані у базу, повинні надсилати свої запити ведучому вузлу, який спочатку записує нові дані у своє локальне сховище.
- Інші репліки називаються веденими вузлами. Кожен раз, коли ведучий вузол записує у своє сховище нові дані, він також надсилає інформацію про зміни даних усім веденим вузлам як частину журналу реплікації або потоку змін. Усі ведені вузли отримують журнал від ведучого та відповідно оновлюють свою локальну копію бази даних, застосовуючи всі операції запису у порядку, в якому вони обробляються ведучим вузлом.
- Коли клієнту потрібно прочитати дані з бази, він може зробити запит або до ведучого вузла, або до будь-якого з ведених. Проте запити на запис дозволяється відправляти тільки ведучому (ведені з точки зору користувача доступні тільки для читання).

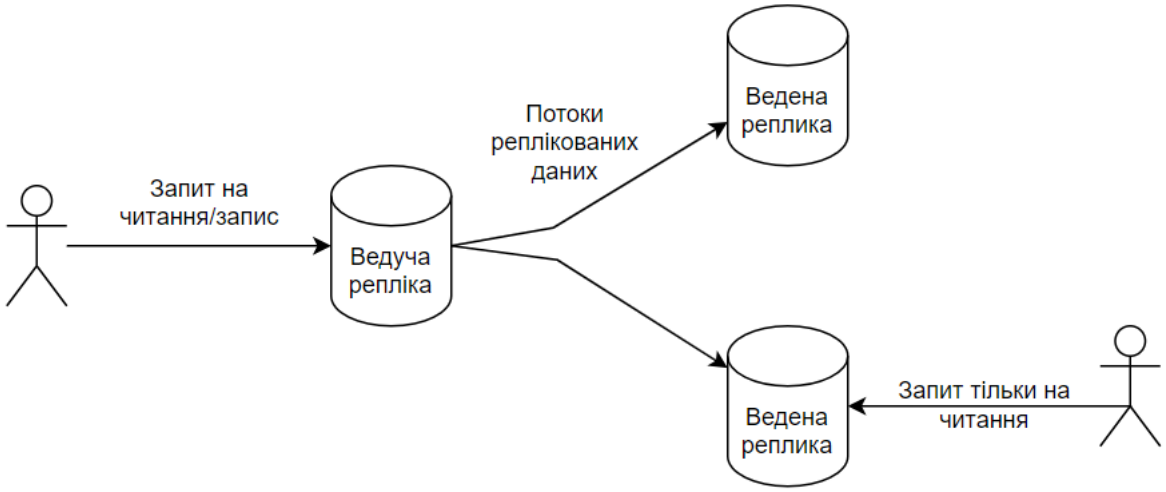

Рис. 4.3. Схема реплікації даних із однією ведучою реплікою та декількоме веденими

Секціонування рекомендоване для реалізації з використанням алгоритму секціонування за хешем ключа. Цей метод полягає в тому, що дані розподіляються

на різні секції чи фрагменти залежно від хеш-значення ключа запису. Хешування ключів гарантує рівномірне розподілення даних між різними секціями.

Використання алгоритму секціонування за хешем ключа має декілька переваг. Воно дозволяє забезпечити рівномірність розподілу навіть при збільшенні обсягу даних. Крім того, цей підхід спрощує пошук записів та знижує конфлікти при доступі до даних. При розробці системи бази даних із секціонуванням за хешем ключа важливо враховувати особливості вибору хеш-функцій та контролювати розмір секцій для оптимального розподілу навантаження. Оптимізація алгоритмів хешування та розмірів секцій може позитивно вплинути на продуктивність та масштабованість системи бази даних.

4.3. Проектування сервісів

Проектування є вирішальним кроком перед реалізацією системи. В цьому підрозділі буде розгорнуто та імплементовано ті модулі, які зображено на діаграмі компонентів. Використаємо метод проектування через диіграму класів, зображену

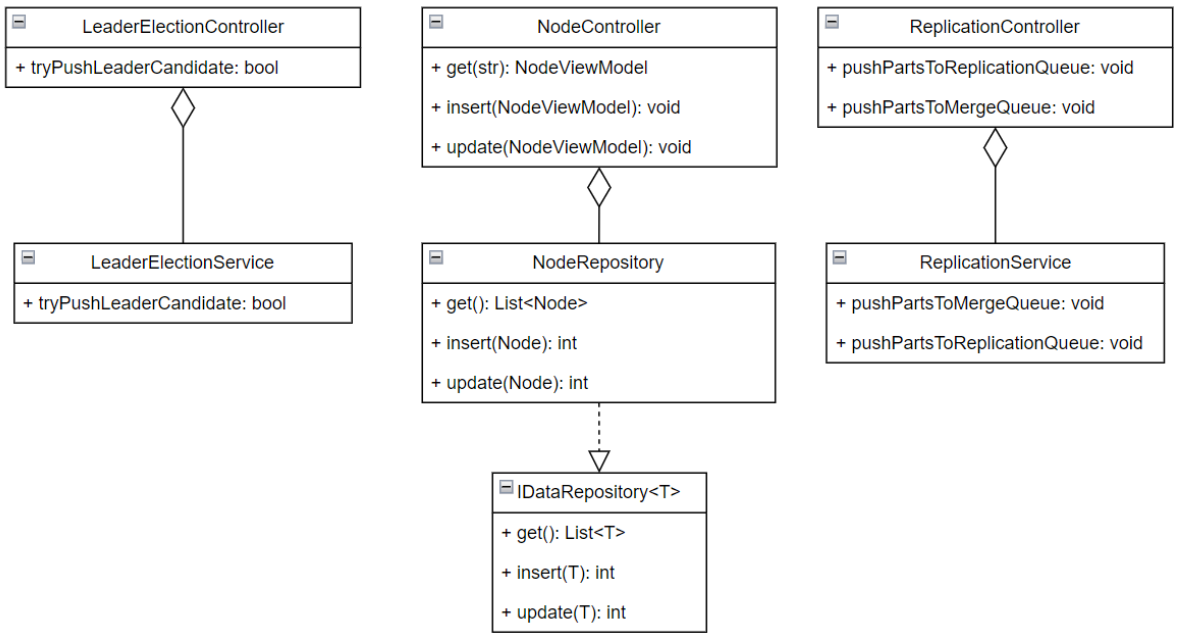

Рис. 4.4. Діаграма класів Меденжера кластера

на Рис. 4.4.

Менеджер кластеру може бути реалізовано як просте сховище типу ключ-значення, цього достатньо для реалізації реплікації між серверами бази даних. Відмінність менеджеру реплікації від звичайного сховища типу ключ-значення полягає в атомарності виконання команд та наявності деяких алгоритмів, наприклад, алгоритму для обирання сервера-лідера реплікації.

Менеджер представлено як Web API сервіс, NodeController якого реалізує набір кінцевих точок, що є доступними для серверів бази даних. NodeController використовує NodeRepository дл зберігання та читання ключів із бази.

Також наявні LeaderElectionController та ReplicationController, які надають серверам бази даних API для обирання лідером реплікації та для відправки лідером повідомлень до решти реплік тієї самої секції (тобто додавання дій до журналу реплікації, розглянутого в підрозділі 4.2.3).

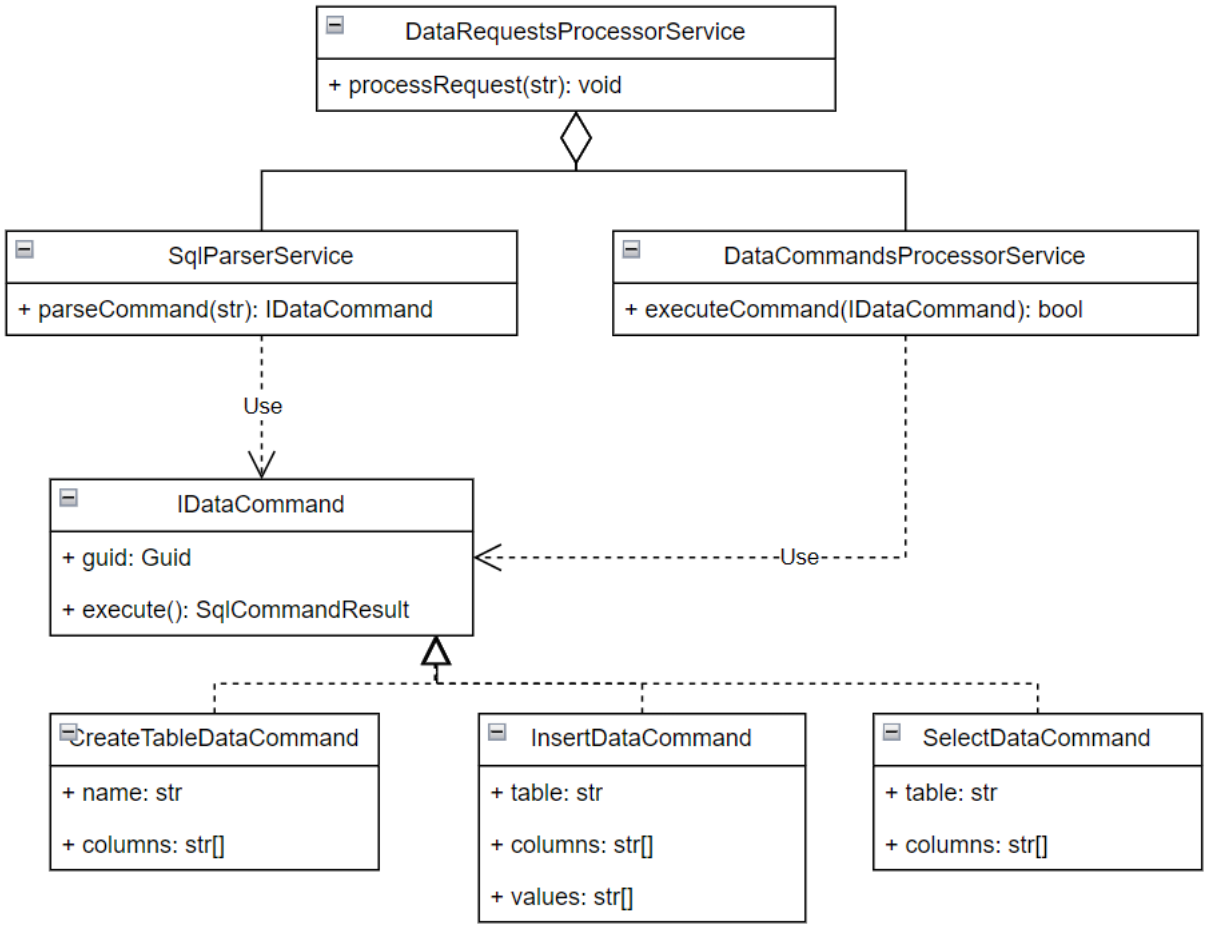

Рис. 4.5. Діаграма класів модуля “Сервіс обробки запитів” компонента “Сервер бази даних”

Сервер бази даних має набагато більше модулів, які також є складнішими за модулі наявні в менеджері реплікації.

Модуль обробки запитів включає в себе SqlParserService, що транслює запити користувача (CREATE…, INSERT…, SELECT…) у команди – IDataCommand, які пізніше обробляються виконувачем команд – DataCommandsProcessorService, що працює стандартизовано, абстрагуючись від деталей конкретної команди, завдяки використанню патерну проектування “Команда”.

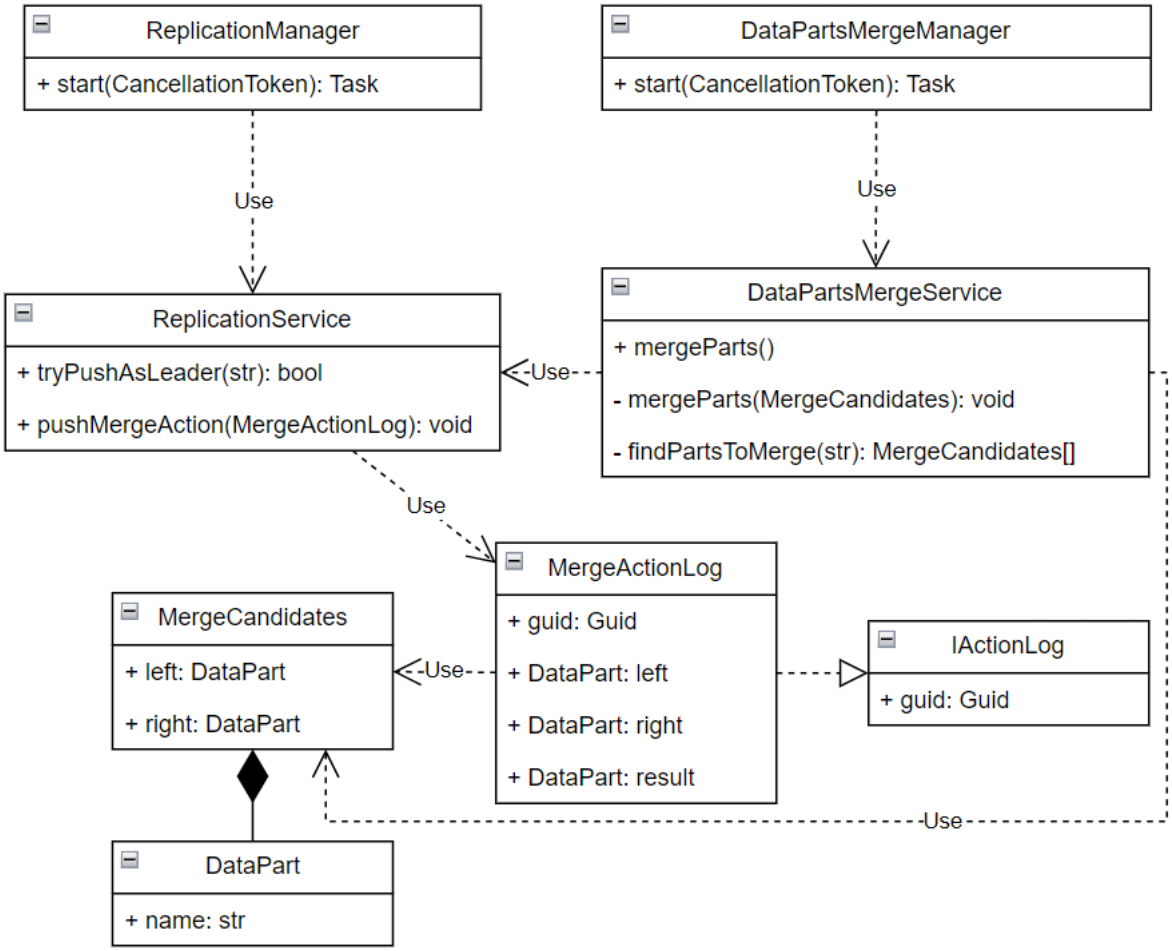

Рис. 4.6. Діаграма класів модулів “Сервіс реплікації” та “Менеджер злиття частин даних” компонента “Сервер бази даних”

Модулі реплікації та персистивного сховища тісно пов'язані, адже реплікацію доцільніше побудувати саме на частинах даних. В порівнянні із реплікацією команд, такий підхід має низку переваг:

- Команди важко піддаються стисканню, отже, при реплікації стиснених частин, відбувається передача набагато меншої кількості даних між репліками.
- Реплікація на базі частин дозволяє не розробляти окремий механізм для синхронізації реплік у разі заміни або тривалого відключення сервера. Адже у такому разі механізм реплікації просто завантажить частини, яких бракує, з інших реплік.

ReplicationService має метод tryPushAsLeader(), цей метод повинен використовуватися перед початком злиття частин (ще одна причина чому модулі реплікації та персистивного сховища тісно пов'язані), адже тільки одна з реплік (один з серверів в секції) можут ьвиконувати злиття частин однієї таблиці в один проміжок часу, для виключення наявності різних наборів злитих частик на різних репліках.

DataPartsMergeService час від часу визначає частини (DataParts), що мають бути злиті (коли кількість малих частин перевищує деякий мінімум), після чого використовує ReplicationService, щоб стати лідером реплікації цієї таблиці. В разі успіху, починається процес злиття частин, після завершення якого визивається метод pushMergeAction(), щоб додати команду злиття цих двох частин в менеджер реплікації, звідки ця команда буде прийнята та виконана усіма іншими репліками.

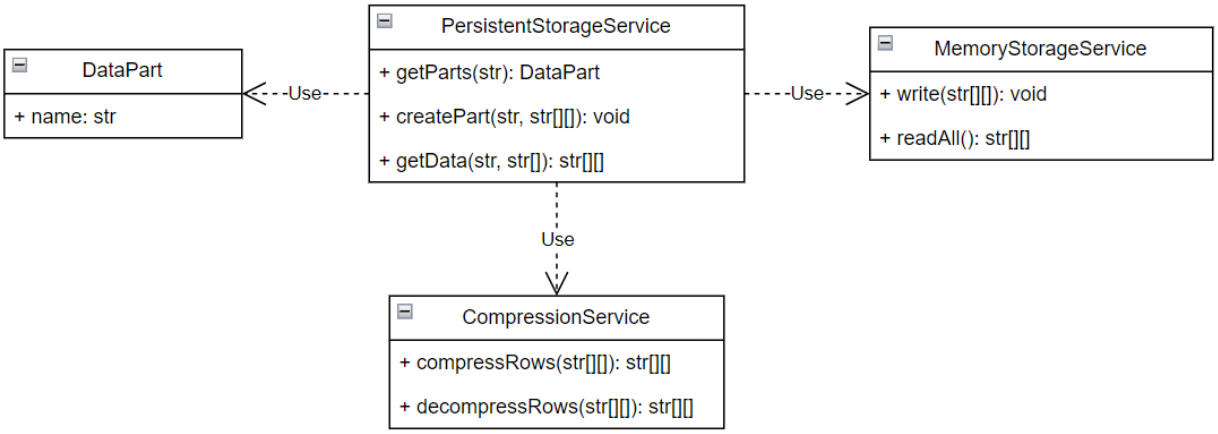

Рис. 4.7. Діаграма класів модуля “Персистивне сховище”

Модуль персистивного сховища також являє собою функціонал, завдяки якому можна отримати список локальних частин даних (getParts()), створити нову частину (createPart()), та отримати дані (getData()). Ці методи використовуються модулем виконання запитів, для виконання вставок (INSERT) та читань (SELECT) даних.

Також наявним є модуль стиснення даних, який використовується перед записом нових частин даних на диск.

4.4. Тестування прототипу

Створений за вищенаведеним проетом прототип системи імплементує вивчені у минулих розділах базові принципи OLAP баз даних, та дозволяє на прикладі розглянути, як вони працюють, та краще зрозуміти їх переваги.

4.4.1. Створення таблиці

Створимо нову таблицю в базі даних:

```
CREATE TABLE products (Id  string, Name string, Description string, price string) PRIMARY KEY Id
```

Після чого в теці \metadata було створено файл products.json, що відображає схему створеної таблиці:

```
{
"guid": "ff3abd4e-d152-4b3c-ae64-be45a6ca9311",
"name": "products",
"columns": [
     {
          "Name": "Id",
          "Type": "string",
          "IsPrimaryKey"
     },
     {
          "Name": "Name",
          "Type": "string",
     },
     {
          "Name": "Description",
          "Type": "string",
     },
     {
          "Name": "Price",
```

```
            "Type": "string",
        }
    ]
    }
```

Далі база баних буде використовувати ці метадані при обробці запитів до цієї таблиці.

4.4.2. Запис даних у таблицю

Додамо нові дані в таблицю:

```
INSERT '200', 'Chair', 'Furniture', '215.15' INTO products
INSERT '58', 'Sofa', 'Furniture', '215.15' INTO products
INSERT '136', 'Bed', 'Furniture', '215.15' INTO products
INSERT '315', 'Mirror', 'Furniture', '215.15' INTO products
```

Такий запит спричинить створення нової теці 1_1_0 в теці \data, де 1 є мінімальним та максимальним номером блока даних, що входить до цієї частини (номер блока даних – це номер, що автоматично збільшується на 1 після кожної вставки даних), а 0 – рівень в дереві злияття. Всередині теці буде створено по файлу для кожного стовпця (адже дані зберігаються по стовпцям): \1_1_0 \Id, \1_1_0 \Name, \1_1_0 \Description, \1_1_0 \Price. Для прикладу, файл \1_1_0 \Id виглядає наступним чином:

```
58
136
200
315
```

Варто зауважити, що строки є відсортованими за первинним ключем, при тому що вставка проводилася в довільному порядку. Саме це є ключовим позитивним фактором у використанні жернальованих дерев: завдяки проведенню даних через бінарне дерево в пам'яті, строки є відсортованими, а запис частини даних було впроведено послідовно, без довільного доступу до диску.

Додамо ще одну партію даних:

```
INSERT '118', 'Writing desk', 'Furniture', '215.15' INTO products
```

```
    INSERT '540', 'Table', 'Furniture', '215.15' INTO products
    INSERT  '34',  'Cabinetry',  'Furniture',  '215.15'  INTO
products
    INSERT '210', 'Chest', 'Furniture', '215.15' INTO products
```

Нова вставка даних призведе до створення це однієї теці в теці \data: 2_2_0. Значення файлу 2_2_0\Id.data виглядатимуть наступним чином:

```
34
118
210
540
```

Після другої вставки, модуль “Менеджер злияття частин даних” помітить дві частини даних, що можна злити в одну задля кращої фрагментації. Результатом цього процесу буде виконання алгоритму злияття та запису його результату у вигляді нової частини даних: 1_2_1. Тобто, нова частина включатиме блоки даних від 1 до 2, та знаходитиметься на другому рівні журнальованого дерева злияття. Файл 1_2_2\Id виглядатиме наступним чином:

```
34
58
118
136
200
210
315
540
```

Завдяки зберіганню даних у відсортованому вигляді, вони можуть бути обїєднані із збереження сортування без додаткових обчислень.

4.4.3. Читання даних з таблиці

Виконаємо запит за значеннями усіх стовпців таблиці products для строки с Id 210:

```
SELECT * FROM products WHERE Id = 210
```

В файлах \1_2_2 \Id, \1_2_2 \Name, \1_2_2 \Description та \1_2_2 \Price строки відсортовані в тому самому порядку, тобто значення строки N з файла одного

стовпця відповідає тій самій строці, що й значення строки N з файла іншого стовпця. Тож, щоб вибрати значення усіх стовпців продукту з Id 210, база спочатку знаходить номер строки з Id 210 у файлі Id.data, а потім зчитує значення на тій самій позиції з усіх інших файлів стовпців.

Так як дані є відсортованими, пошук в файлі \1_2_2 \Id може бути виконаним за алгоритмом бінарного пошуку. База даних отримає відступ потрібної строки у файлах рядків, зчитає значення 6-ї строки з кожного файлу стовпців, та отримає строку результату:

```
210, Chest, Furniture, 215.15
```

4.5. Висновки

Прототипування є важливим етапом дослідження методів розробки програмного забезпечення, адже дозволяє довести можливість реалізації алгоритмів, побачити практичні проблеми на ранньому етапі із ціллю покращення теоретичного материалу перед реалізацією повноцінного програмного забезпечення, а також протестувати базові модулі програмного забезпечення, в тому числі під навантаженням.

При цьому, прототипуванню передує проектування програмного забезпечення, що й було зроблено: цей розділ включає опис та розробку компонентів системи: Менеджера кластера і Сервера бази даних та модулів, з яких вони складаються. Далі ці модулі було розглянуто більш детально завдяки діаграмам класів, які вже, в свою чергу, надають знання щодо методологочних аспектів імплементації запланованого функціоналу.

Результатом цього розділу є мінімальний життєздатний продукт, що доводить можливості реалізації проекту, а також надає методологічні вказівки та рекомендації до реалізації подібних систем.

ВИСНОВКИ

В рамках даного проекту було спроектовано розподілену аналітичну базу даних, включаючи обидва компоненти кластера: Менеджер кластера та Сервер бази даних, а також було реалізовано прототип системи, що надає можливість розглянути методологію розробки подібних систем на прикладі, включаючи рекомендації з використання та імплементації алгоритмів.

Метою цього проекту було дослідження сучасних вимог до сховищ великих даних, огляд теорії та формування методологічних засад розробки таких систем. Для досягнення цієї мети були поставлені і реалізовані завдання по дослідженню історичного контексту, основних теоретичних концепцій, існуючих розробок, вимог до OLAP баз даних, а також алгоритмів та структур даних. Окрім того, було сформовано ТЗ до розробки такої системи.

В ході аналізу предметної області, було виявлено, що сучасні обсяги інформації вимагають від компаній, що працюють із великою кількістю користувачів, використання нових технологій, які дозволяють такі обсяги зберігати та, найважливіше, обробляти. Було сформульовано практичну значимість OLAP баз даних, а саме – виявлено бажання бізнесу аналізувати дані з дій, придбання товарів, мереміщень, тощо, із метою швидкого реагування на зміни ринку в умовах конкуренції. Як було з'ясовано, такі вимоги задовольняються OLAP базами даних, що є розподіленими, меньш зав'язаними на транзакційність та консистентність даних, що надає перевагу у гнучкості використання та швидкості.

Особливий наголос цей проект робить на Розділі 3, який описує процес дослідження алгоритмів, що використовуються в OLAP базах даних. Цей розділ має особливу значимість з точки зору популяризації та надання методологіних вказівок для розробки подібних систем, адже в ному розглянуто три основні алгоритми, що наділяють OLAP бази даних дослідженими перевагами над класичними транзакційними базами даних: журнальовані дерева злияття, стовпчикове зберігання даних із стисненням та розподілення (реплікація та секціонування) даних. Цей розділ рекомендується для ознайомлення розробникам,

що мають базові знання про класичні бази даних, та мають бажання дослідити можливості OLAP баз даних, адже алгоритми розглянуто у порівнянні: журнальовані дерева порівнюються із класичною реалізацієї індекса – бінарним деревом, стовпчикове зберігання – із рядковим. Окрім того, кожна реалізація алгоритмів була протестована під навантаженням, для порівняння продуктивності класичних реалізацій із такими, що використовуються в OLAP базах даних.

Завершальним етапом цього проекту є проектування OLAP бази даних із використанням діаграм компонентів, класів, та довільних візуалізацій алгоритмів, із подальшою реалізацією прототипу системи. Розроблений прототип було розглянуто на практиці, тож принципи, на яких він побудований, було розглянуто наочно.

Підсумовуючи вищезгадане, можна впевнено стверджувати, що даний дипломний проект успішно досягнув своєї мети та вирішив поставлені завдання. Проведено детальний аналіз OLAP систем управління базами даних, було описано теоретичні та методологічні аспекти їх розробки. Отримані знання та досвід виявляться корисними при подальших дослідженнях в цій області та для практичного застосування у розробці OLAP систем.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Brewer E. A., Hellerstein J. M. CS262a: Advanced Topics in Computer Systems // lecture notes, University of California, Berkeley, August 2011 [Електронний ресурс]. — Режим доступу до ресурсу: http://people.eecs.berkeley.edu/~brewer/cs262/systemr.html.
2. Li, Y., He, B., Yang, R.J., Luo, Q. and Yi, K.. Tree indexing on solid state drives. Proceedings of the VLDB Endowment, 2010, С. 1195-1206.
3. Graefe G. Modern B-Tree Techniques // Foundations and Trends in Databases, volume 3, number 4, pages 203–402, August 2011 [Електронний ресурс]. — Режим доступу до ресурсу: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.219.7269& rep=rep1&type=pdf.
4. O'Neil, P., Cheng, E., Gawlick, D., & O'Neil, E. (1996). The log-structured merge-tree (LSM-tree). Acta Informatica, 33, 351-385.
5. Zaitsev P. Innodb Double Write. August 4, 2006 [Електронний ресурс]. — Режим доступу до ресурсу: https://www.percona.com/blog/2006/08/04/innodb-double-write/.
6. Callaghan M. The Advantages of an LSM vs a B-Tree. January 19, 2016 [Електронний ресурс]. — Режим доступу до ресурсу: https://smalldatum.blogspot.com/2016/01/summary-of-advantages-of-lsm-vs-b-tree.html.
7. Boncz P., Zukowski M., Nes N. MonetDB/X100: Hyper- Pipelining Query Execution//2nd Biennial Conference on Innovative Data Systems Research (CIDR), January2005 [Електронний ресурс]. — Режим доступу до ресурсу: http://www.cidrdb.org/cidr2005/papers/P19.pdf.
8. Zhou J., Ross K. A. Implementing Database Operations Using SIMD Instructions // ACM International Conference on Management of Data (SIGMOD), pages 145–156, June 2002 [Електронний ресурс]. — Режим доступу до ресурсу: http://www1.cs.columbia.edu/~kar/pubsk/simd.pdf.

9. Abadi D. J., Boncz P., Harizopoulos S., et al. The Design and Implementation of Modern Column-Oriented Database Systems // Foundations and Trends in Databases, volume 5, number 3, pages 197–280, December 2013 [Електронний ресурс]. — Режим доступу до ресурсу: http://cs-www.cs.yale.edu/homes/dna/papers/abadi-column-stores.pdf

10. DeWitt D. J., Gray J. N. Parallel Database Systems: The Future of High Performance Database Systems // Communications of the ACM, volume 35, number 6, pages 85–98, June 1992 [Електронний ресурс]. — Режим доступу до ресурсу: https://15799.courses.cs.cmu.edu/fall2013/static/papers/dewittgray92.pdf