НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Інститут телекомунікаційних систем

Кафедра Інформаційно-телекомунікаційних мереж

До захисту допущено:

В.о. завідувача кафедри

_____ Лариса ГЛОБА

«___»_____2021 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інформаційно-комунікаційні технології»

спеціальності 172 «Телекомунікації та радіотехніка»

на тему: «Аналіз платформ зберігання великих даних»

Виконала:
студентка IV курсу, групи ТІ-71
Перебаєва Катерина Сергіївна _____

Керівник:
професор кафедри ІТМ, д.т.н., с.н.с.
Скулиш Марія Анатоліївна _____

Рецензент:
професор кафедри ТК, д.т.н., с.н.с.
Лисенко Олександр Іванович _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2021 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Інститут телекомунікаційних систем
Кафедра Інформаційно-телекомунікаційних мереж

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 172 «Телекомунікації та радіотехніка»

Освітньо-професійна програма «Інформаційно-комунікаційні технології»

ЗАТВЕРДЖУЮ
В.о.завідувача кафедри
_____ Лариса ГЛОБА
«___»_____2021 р.

ЗАВДАННЯ
на дипломну роботу студенту
Перебаєвої Катерини Сергіївни

1. Тема роботи «Аналіз платформ зберігання великих даних», керівник роботи Скулиш Марія Анатоліївна, професор кафедри інформаційно-телекомунікаційних мереж ІТС, професор, д.т.н., с.н.с,, затверджені наказом по університету від «14» квітня 2021 р. № 1007-с

2. Термін подання студентом роботи 7 червня 2021 р.

3. Вихідні дані до роботи: теоретичні дані, документація

4. Зміст роботи:

4.1. Актуальність задачі, огляд існуючих підходів

4.2.Дослідження та аналіз можливих рішень NoSQL

4.3. Практичне застосування на прикладі роботи з медичними даними

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)

6. Дата видачі завдання 25 жовтня 2020 року

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Огляд існуючих рішень за темою роботи	30.11.2020	Виконано
2	Дослідження актуальності обраної теми	10.01.2021	Виконано
3	Розробка інформаційної моделі для медичних карт	12.02.2021	Виконано
4	Аналіз видів не реляційних моделей даних	25.03.2021	Виконано
5	Вибір оптимальної моделі даних для медичних досліджень	20.04.2021	Виконано
6	Вибір платформи зберігання великих даних	24.05.2021	Виконано

Студент Катерина ПЕРЕБАЄВА

Керівник Марія СКУЛИШ

РЕФЕРАТ

Дипломна робота: 94с., 16 рис., 2 табл., 51джерело.

Мета дослідження: забезпечити доступність, масштабованість, швидкість аналізу Великих даних, за рахунок оптимального вибору системи зберігання даних, яка дозволить надати якісні послуги кінцевим користувачам.

У даній роботі проаналізувано та надано класифікацію типів не реляційних баз даних, проаналізовано платформи зберігання великих даних

Розроблено інформаційну модель даних медичних карт та вибрано оптимальну платформу зберігання великих даних для практичного завдання на прикладі роботи с медичними даними.

Описано структуру бази даних для медичних карт в порядку розгортання від основних документів до побічних.

Ключові слова: ВЕЛИКІ ДАНІ, НЕРЕЛЯЦІЙНІ БАЗИ ДАНИХ, КЛАСИФІКАЦІЯ, АНАЛІЗ.

ABSTRACT

Bachelor`s thesis: 94p., 16fig., 2 tabl., 51 sources.

Purpose of research: ensure the availability, scalability, speed of analysis of Big Data, through the optimal choice of storage system that will provide quality services to end users.

This work analyzes and provides a classification of types of non-relational databases, analyzes big data storage platforms.

Developed an information model of medical card data and selects the optimal big data storage platform for a practical task on the example of working with medical data.

The structure of the database for medical card is described in the order of deployment from the main documents to the secondary ones.

Key words: BIG DATA, TELECOMMUNICATIONS, NOSQL, CLASSIFICATION, ANALYSIS.

ЗМІСТ

Y

ПЕРЕЛІК СКОРОЧЕНЬ

БД	База даних
ПЗ	Програмне забезпечення
ІТ	Інформаційні технології
NoSQL	Not Only SQL
БІ	Бізнес-аналітика
IoT	Інтернет речей
СУБД	Система управління базами даних
URI	Уніфікований ідентифікатор ресурсу

ВСТУП

Актуальність теми

Сучасний світ – це величезний цифровий простір. Ми управляємо, ділимося та зберігаємо всі наші аспекти життя онлайн. У кожного з нас є акаунт в соціальних мережах, де ми ділимося фото, відео та будь-якою інформацією про наше життя. Дані з усіх наших пристроїв – комп'ютерів, планшетів, смартфонів – постійно збираються та передаються в мережу. Крім того, ми ще робимо покупки онлайн, користуємося пошуком, завантажуємо музику, тобто самостійно продукуємо гігабайти інформації. Обсяги даних заворожують дух.

Тільки уявіть, рівень користувачів інтернету зараз становить 59,5%. Однак COVID-19 значно вплинув на збір даних про кількість користувачів інтернету, тому фактичні цифри можуть бути вище. А соціальними мережами в 2021 році користуються 53,6% світового населення.

Цікавий факт: якщо зібрати всю інформацію, яку людство накопичило з початку до 2000 року включно, то виявиться, що це менше, ніж ми виробляємо за одну хвилину. Це явище повністю змінює уявлення про світ і наше місце в ньому.

Обсяг зібраних даних зростає величезними темпами.

Разом зі стрімким накопиченням інформації швидкими темпами розвиваються і технології аналізу даних. Якщо ще нещодавно було можливо лише сегментувати клієнтів на групи зі схожими перевагами, то тепер можливо будувати моделі для кожного клієнта в режимі реального часу, аналізуючи, як приклад, його переміщення по мережі Інтернет для пошуку конкретного товару. Інтереси споживача можуть бути проаналізовані, і відповідно до побудованої моделі виведена відповідна реклама або конкретні пропозиції. Модель також може налаштовуватися і перебудовуватися в режимі реального часу.

Застосування Big Data цікаво не тільки з метою маркетингових досліджень, а й для аналізу недоотриманого прибутку. Організація може отримати переваги від застосування технологій великих даних, якщо її існуючі програми і бази даних більше не здатні масштабируватися і справлятися з раптовими збільшеннями обсягу або різноманітності даних або вимог до швидкості їх обробки. Якщо вчасно не знайти правильний підхід до роботи з великими даними, це може привести до підвищення витрат, а також зниження ефективності роботи і конкурентоспроможності.

Big Data – це одночасно і великі можливості, і великі проблеми.

Об’єми великих даних значно збільшуються із року в рік. Великі компанії, які оперують великими даними, мають не лише їх накопичувати, а й зберігати, швидко і точно обробляти, щоб отримати користь. Це вимагає принципово нових пристроїв і алгоритмів для зберігання інформації.

Об’єкт дослідження

Процес структурування та організації великих даних, моделі та методи функціонування платформ зберігання та обробки великих даних.

Предмет дослідження

Сучасні платформи зберігання великих даних.

Мета дослідження

Забезпечити доступність, масштабованість, швидкість аналізу Великих даних, за рахунок оптимального вибору системи зберігання даних, яка дозволить надати якісні послуги кінцевим користувачам.

Задачі дослідження

- Проаналізувати платформи зберігання великих даних за різними критеріями.

- Виявити плюси та мінуси різних платформ

- Розробка математичної моделі даних медичних карт

- Вибір оптимальної платформи зберігання великих даних для практичного завдання

РОЗДІЛ 1
АКТУАЛЬНІСТЬ ЗАДАЧІ, ОГЛЯД ІСНУЮЧИХ ПІДХОДІВ

1.1 Визначення великих даних

Термін Big Data з'явився в 2008 році. Вперше його вжив редактор журналу Nature - Кліффорд Лінч. Він розповідав про вибухове зростання обсягів світової інформації і відзначав, що освоїти їх допоможуть нові інструменти і більш розвинені технології.

З назви можна припустити, що термін «Великі дані» відноситься просто до управління та аналізу великих обсягів даних. Згідно зі звітом McKinsey Institute «Великі дані: новий рубіж для інновацій, конкуренції і продуктивності» (Big data: The next frontier for innovation, competition and productivity), термін «Великі дані» відноситься до наборів даних, розмір яких перевищує можливості типових баз даних (БД) по занесенню, зберігання, управлінню та аналізу інформації. І світові репозиторії даних, безумовно, продовжують рости.

«Великі дані» припускають щось більше, ніж просто аналіз величезних обсягів інформації.

На сьогодні поняття «Великі дані» (Big Data) є одним з найбільш часто використовуємих у корпоративному середовищі, але не має точного визначення і меж цього поняття. Безліч авторів, організацій та компаній намагаються по-різному його інтерпретувати.
Що ж таке Big Data?

Big Data - це серія підходів, інструментів і методів, використовуваних для обробки структурованих і неструктурованих даних величезних обсягів і значного різноманіття для отримання результатів, які сприймаються людьми,

які доводять свою ефективність в умовах безперервного зростання. Великі дані служать альтернативою традиційним системам управління базами даних і рішень в рамках Business Intelligence. [1.]

Найбільш популярним вважається визначення від Forrester Research (2013): «Великі дані представляють масивний об'єм структурованих і не структурованих даних, які настільки великі, що їх складно обробляти, використовуючи традиційні бази даних і програмні засоби». [2] Ефективність такого визначення, що з'явилося в той час, коли з'явилися перші засоби для роботи з великими даними і намітилися певні тенденції в області розробки і застосування великих даних, в основному пов'язана з розробкою нереляційних баз даних (NoSQL), засобів зберігання великих даних і масштабованих засобів масивної обробки, таких як MapReduce, Hadoop, Spark.

Як бачимо, в цьому визначенні присутні такі невизначені терміни, як «величезних», «значного», «ефективно» і «масивних». Навіть сама назва вельми суб'єктивно.

Аналітики компанії IBS «весь світовий обсяг даних» оцінили такими величинами:

2003 г. - 5 ексабайт даних (1 ЕБ = 1 млрд гігабайтів)

2008 - 0,18 зеттабайт (1 ЗБ = 1024 ексабайта)

2015 г. - понад 6,5 зеттабайт

2020 г. - 40-44 зеттабайт (прогноз)

2025 г. - цей обсяг зросте ще в 10 разів. [3]

У вересні 2020 року агентство ResearchAndMarkets опублікувало звіт про глобальний ринок аналітики великих даних. Згідно з опублікованою інформацією, світовий ринок аналітики великих даних оцінюється в $ 41,85 млрд у підсумками 2019 року. За прогнозами аналітиків, він виросте до $ 115,13 млрд, де середній динаміці в 11,9% протягом прогнозованого періоду з 2020 по 2028 рік.

Як наголошується в звіті, аналітика великих даних може бути названа серцем цифрового світу, на підставі аналізу та перетворення даних в інформацію, яка надає цінні ідеї для бізнесу. Хмарні платформи є основними для додатків аналізу великих даних. При цьому для проведення аналізу великих даних великі організації в основному використовують гібридну хмарну платформу, в той час як публічні хмари переважають серед малих і середніх організацій.

1.2. Характеристики та завдання великих даних

Великі дані мають специфічні характеристики та властивості, які можуть допомогти вам зрозуміти як виклики, так і переваги ініціатив у галузі великих даних.

У 2001 році вийшло основоположне дослідження Gartner, яке визначило три основні характеристики великих даних: обсяг (в сенсі фізичного обсягу), швидкість (як швидкість приросту, так і швидкості необхідної обробки та отримання результатів) і різноманітність даних (в сенсі можливості одночасної обробки різних типів структурованих і слабоструктурованих даних) (англ. – volume, velocity, variety).

Надалі до базових 3V додалися властивості великих даних, які вони набувають в процесі обробки і які є основоположними для визначення вимог до систем обробки великих даних: Value, Variability, Veracity (Значимість, Мінливість, Конфіденційність). Зараз їх набагато більше.

Ось 10V великих даних:

Об'єм:

Обсяг - це, мабуть, найвідоміша характеристика великих даних, це не дивно, оскільки понад 90 відсотків усіх сьогоднішніх даних було створено за останні пару років. Поточний обсяг даних насправді може бути приголомшливим.

Швидкість:

Швидкість означає швидкість, з якою дані генеруються, виробляються, створюються або оновлюються. Звичайно, вражає, що сховище даних Facebook зберігає понад 300 петабайт даних, але слід враховувати швидкість створення нових даних. Facebook вимагає 600 терабайт вхідних даних на день. Тільки Google обробляє в середньому понад "40 000 пошукових запитів щосекунди", що приблизно означає понад 3,5 мільярда пошукових запитів на день.

Різновид:

Що стосується великих даних, ми маємо обробляти не тільки структуровані дані, але й напівструктуровані та переважно неструктуровані дані. Як випливає з вищенаведених прикладів, більшість великих даних здаються неструктурованими, але крім аудіо, зображень, відеофайлів, оновлень соціальних мереж та інших текстових форматів існують також файли журналів, дані кліків, дані машини та датчика тощо.

Варіабельність:

Варіативність у контексті великих даних стосується кількох різних речей. Одне - кількість невідповідностей даних. Їх потрібно знайти за допомогою методів виявлення аномалій та нестабільних ситуацій, щоб відбулася будь-яка значуща аналітика.

Великі дані також є змінними через безліч вимірів даних, що виникають внаслідок різних різнорідних типів даних та джерел. Варіативність може також стосуватися непослідовної швидкості завантаження великих даних у вашу базу даних.

Правдивість:

Це одна з невдалих характеристик великих даних. У міру збільшення будь-якого або всіх вищезазначених властивостей правдивість (довіра чи довіра до даних) падає. Це схоже на, але не те саме, що дійсність чи волатильність. Достовірність стосується більше походження або надійності

джерела даних, його контексту та того, наскільки значущим він є для аналізу на його основі.

Наприклад, розглянемо набір статистичних даних про те, що люди купують у ресторанах, та ціни на ці товари за останні п’ять років. Ви можете запитати? Хто створив джерело? Якою методологією вони керувались при зборі даних? Чи були включені лише певні кухні чи певні типи ресторанів? Чи узагальнили інформацію творці даних? Чи редагував чи модифікував інформацію хтось інший?

Відповіді на ці запитання необхідні для визначення достовірності цієї інформації. Знання достовірності даних, у свою чергу, допомагає нам краще зрозуміти ризики, пов’язані з аналізом та діловими рішеннями на основі цього конкретного набору даних

Термін дії:

Подібно до правдивості, достовірність означає, наскільки точними та правильними є дані для їх використання за призначенням. Час спеціаліста з обробки даних витрачається на очищення даних, перш ніж мати можливість провести будь-який аналіз. Вигода від аналітики великих даних настільки ж хороша, як і базові дані, тому вам потрібно застосувати належні практики управління даними, щоб забезпечити стабільну якість даних, загальні визначення та метадані.

Вразливість:

Великі дані викликають нові проблеми безпеки. Зрештою, порушення даних із великими даними - це велике порушення. На жаль, було багато порушень великих даних. Ще один приклад, як повідомляв CRN у травні 2016 року, "хакер під назвою Peace розмістив дані у темній мережі для продажу, які нібито включали інформацію про 167 мільйонів облікових записів LinkedIn та 360 мільйонів електронних листів та паролів для користувачів MySpace".

Волатильність:

Скільки років має бути вашим даним, перш ніж вони вважатимуться неактуальними, історичними чи більше не корисними? Як довго потрібно зберігати дані?

До великих даних організації, як правило, зберігали дані необмежено довго, кілька терабайт даних можуть не створювати великих витрат на зберігання; його можна навіть зберігати в базі даних, не викликаючи проблем із продуктивністю. У класичній обстановці даних може навіть не існувати політики архівування даних.

Однак через швидкість та обсяг великих даних, їх мінливість потрібно ретельно враховувати. Тепер вам потрібно встановити правила щодо валюти та доступності даних, а також забезпечити швидкий пошук інформації, коли це потрібно. Переконайтеся, що вони чітко пов’язані з вашими бізнес-потребами та процесами з великими даними, витрати та складність процесу зберігання та пошуку збільшуються.

Візуалізація:

Ще однією характеристикою великих даних є те, наскільки складно їх візуалізувати. Поточні засоби візуалізації великих даних стикаються з технічними проблемами через обмеження технології в пам'яті та погану масштабованість, функціональність та час відгуку. Ви не можете покластися на традиційні графіки, намагаючись побудувати мільярд точок даних, тому вам потрібні різні способи подання даних, такі як кластеризація даних або використання деревних карт, спалахів сонця, паралельних координат, кругових мережевих діаграм або дерев конусів.

Поєднавши це з безліччю змінних, що виникають внаслідок різноманітності та швидкості великих даних та складних взаємозв’язків між ними, ви зможете побачити, що розробити значущу візуалізацію непросто.

Значення:

Останнє, але, мабуть, найважливіше з усіх - це цінність. Інші характеристики великих даних безглузді, якщо ви не отримуєте ділову цінність із даних. У великих даних можна знайти значну цінність,

включаючи краще розуміння своїх клієнтів, відповідну націленість на них, оптимізацію процесів та покращення продуктивності машин або бізнесу. Вам слід зрозуміти потенціал, поряд із більш складними характеристиками, перш ніж приступати до стратегії великих даних.

Цей список можна продовжувати, однак будь-яка проблема - зворотня сторона можливостей.

Завдання, пов'язані з Big Data. Існують три типи завдань пов'язаних з Big Data:

1. Зберігання і управління

Обсяг даних в сотні терабайт або петабайт не дозволяє легко зберігати і управляти ними за допомогою традиційних реляційних баз даних. Big Data зазвичай зберігаються і організовуються в розподілених файлових системах.

У загальних рисах, інформація зберігається на декількох (іноді тисячах) жорстких дисках, на стандартних комп'ютерах. Так звана «карта» (map) відстежує, де (на якому комп'ютері і / або диску) зберігається конкретна частина інформації. Для забезпечення відмовостійкості та надійності, кожну частину інформації зазвичай зберігають кілька разів, наприклад - тричі.

Так, наприклад, припустимо, що ви зібрали індивідуальні транзакції у великій роздрібній мережі магазинів. Детальна інформація про кожну транзакції буде зберігатися на різних серверах і жорстких дисках, а «карта» (map) індексує, де саме зберігаються відомості про відповідну угоду.

За допомогою стандартного устаткування і відкритих програмних засобів для управління цією розподіленою файловою системою (наприклад, Hadoop), порівняно легко можна реалізувати надійні сховища даних в масштабі петабайт.

2. Неструктурована інформація

Більшість всіх даних Big Data є неструктурованими. Тобто як можна організувати текст, відео, зображення, і т.д.?

Це має свої переваги і недоліки.

Перевага полягає в тому, що можливість зберігання великих даних дозволяє зберігати "всі дані", не турбуючись про те, яка частина даних актуальна для подальшого аналізу і прийняття рішення.

Недоліком є те, що в таких випадках для отримання корисної інформації потрібно подальша обробка цих величезних масивів даних.

Хоча деякі з цих операцій можуть бути простими (наприклад, прості підрахунки, і т.д.), інші вимагають більш складних алгоритмів, які повинні бути спеціально розроблені для ефективної роботи на розподіленої файлової системи.

3. Аналіз Big Data

Як аналізувати неструктуровану інформацію? Як на основі Big Data складати прості звіти, будувати і впроваджувати поглиблені прогностичні моделі?

1.3. Визначення платформи великих даних

Платформа великих даних – це інструмент, розроблений постачальниками програмного забезпечення (ПЗ) для управління даними з метою покращення масштабованості, доступності, продуктивності та безпеки організацій, які працюють з великими даними.

Платформа великих даних, як правило, складається із сховища великих даних, серверів, бази даних, управління великими даними, бізнес-аналітики та інших утиліт управління великими даними. Він також підтримує розробку, запити та інтеграцію з іншими системами.

Основною перевагою платформи великих даних є зменшення складності декількох постачальників / рішень в єдине ціле рішення. Платформа великих даних також постачається через хмару, де провайдер надає комплексні рішення та послуги для великих даних.

Платформа великих даних також зосереджена на наданні своїм користувачам ефективних інструментів аналітики для масивних наборів даних. Ці платформи часто використовуються інженерами даних для збору, очищення та підготовки даних для бізнес-аналізу. Вчені данних використовують цю платформу для виявлення взаємозв'язків та закономірностей у великих наборах даних за допомогою алгоритму машинного навчання. Користувач таких платформ може створювати додатки на власний розсуд, наприклад, розраховувати лояльність клієнтів (електронна комерція), і так далі, існує безліч випадків використання.

Це платформа інформаційних технологій (IT) класу підприємства, яка забезпечує властивості та функціональність прикладної системи в одному рішенні для розробки, розгортання, обробки та управління великими даними. Програмне забезпечення (ПЗ) аналітики великих даних допомагає розкрити приховані шаблони, невідомі кореляції, ринкові тенденції, вподобання клієнтів та іншу корисну інформацію з широкого різноманіття наборів даних.

Головне питання організації роботи з великими даними на корпоративному рівні: обрати реляційну (SQL) чи нереляційну (NoSQL) базу даних? Головною причиною відмови від SQL баз даних (БД) є не правильна робота з самою базою. Більшість компаній не можуть собі дозволити тримати спеціалістів для постійного налагодження баз даних, а для того, щоб розпочати використовувати NoSQL БД не потрібно додаткових розробок. При розробці NoSQL БД особлива увага приділяється забезпеченню високої масштабованості та гнучкості рішень. NoSQL БД – це, перш за все, швидкий доступ до даних, що зберігаються в оперативній пам'яті, гнучкість використання та можливість швидкого розподілення даних між вузлами. Однак можливі такі сценарії, коли дані згодом виходять з-під контроля або вже просто не вміщуються в оперативній пам'яті.

1.4. Основні відомості про зберігання великих даних

Технології зберігання великих даних називаються технологіями зберігання, які певним чином вирішують обсяг, швидкість або різноманітність і не потрапляють до категорії реляційних систем баз даних. Це не означає, що реляційні системи баз даних не вирішують ці проблеми, але альтернативні технології зберігання, такі як стовпчасті магазини та розумні комбінації різних систем зберігання, наприклад використання HadoopDistributed File System (HDFS), часто є більш ефективними та менш дорогими (Marzand Warren 2014).

Великі системи зберігання даних, як правило, вирішують проблему обсягу, використовуючи розподілену, спільну архітектуру. Це дозволяє задовольнити підвищені вимоги до зберігання за допомогою масштабування до нових вузлів, що забезпечують обчислювальну потужність та зберігання. Нові машини можна легко додавати до кластера пам’яті, а система зберігання піклується про прозорий розподіл даних між окремими вузлами. Рішення для зберігання також повинні впоратися зі швидкістю та різноманітністю даних. Швидкість важлива з точки зору затримки запиту, тобто як довго чи потрібно відповідь на запит? Це особливо важливо в умовах високих показників надходження даних. Наприклад, випадковий доступ до бази даних може значно уповільнити продуктивність запитів, якщо йому потрібно надати гарантії транзакцій. На відміну від різноманітності, різноманітність відноситься до рівня зусиль, необхідних для інтеграції та роботи з даними, що походять із великої кількості різних джерел. Наприклад, графічні бази даних є придатними системами зберігання для вирішення цих проблем.

В результаті аналізу поточних і майбутніх технологій зберігання даних було отримано ряд уявлень, що стосуються технологій зберігання даних. Стало очевидним, що сховище великих даних перетворилося на товарний бізнес, а масштабовані технології зберігання даних досягли рівня

підприємництва, який може управляти практично необмеженими обсягами даних. Свідченням цього є широке використання рішень на основі Hadoop, пропонованих постачальниками, такими як Cloudera (2014a), Hortonworks (2014) та MapR (2014), а також різними постачальниками баз даних NoSQL2, зокрема тими, що використовують технології пам'яті та стовпчасті сховища. Порівняно з традиційними реляційними системами управління базами даних, які покладаються на швидке зберігання та дорогі стратегії кешування, ці нові технології зберігання великих даних пропонують кращу масштабованість при меншій складності операцій та витратах. досі значний невикористаний потенціал для технологій зберігання великих даних як для використання, так і для подальшого розвитку технологій:

- Потенціал для трансформації суспільства та бізнесу в різних секторах: Технології великого сховища даних є ключовим фактором, що сприяє передовій аналітиці, яка має потенціал для трансформації суспільства та способу прийняття ключових рішень у сфері бізнесу. Хоча ці сектори стикаються з такими нетехнічними проблемами, як відсутність кваліфікованих експертів з великих даних та регуляторні бар'єри, нові технології зберігання даних мають потенціал для створення нової аналітики, що створює цінність, у різних галузях промисловості та між ними.
- Відсутність стандартів є основною перешкодою : Історія NoSQL базується на вирішенні конкретних технологічних проблем, що призводять до цілого ряду різних технологій зберігання. Широкий вибір варіантів у поєднанні з відсутністю стандартів для запитів даних ускладнює обмін сховищами даних, оскільки це може прив'язати специфічний код програми до певного рішення для зберігання даних.
- Відкриті проблеми масштабованості в сховищах даних на основі графіків: Обробка даних, що базуються на графічних структурах даних вигідний у зростаючій кількості заявок. Це дозволяє краще вловлювати семантику та складні взаємозв'язки з іншими частинами інформації, що надходить із великої кількості різноманітних джерел даних, і має потенціал

для покращення загальної цінності, яка може бути отримана шляхом аналізу даних. Хоча для цього все частіше використовуються бази даних графіків, залишається жорстким ефективно розподілити структуру даних на основі графіків між обчислювальними вузлами.

- Конфіденційність та безпека відстає: Хоча існує кілька проектів та рішень, які стосуються конфіденційності та безпеки, захисту індивідуумів та захисту їх дані відстають від технологічного прогресу систем зберігання даних. Потрібні значні дослідження, щоб краще зрозуміти, як можна зловживати даними, як їх потрібно захищати та інтегрувати в рішення для зберігання великих даних.

1.5. Методи зберігання великих даних

В даний час існує два добре усталених способи зберігання великих даних:

Складське зберігання - Подібно до складу для зберігання фізичних товарів, сховище даних - це великий будівельний об’єкт, основною функцією якого є зберігання та обробка даних на рівні підприємства. Це важливий інструмент для аналізу великих даних. Ці великі сховища даних підтримують різні звіти, бізнес-аналітику (BI), аналітику, видобуток даних, дослідження, кібермоніторинг та інші пов’язані з цим види діяльності. Ці склади зазвичай оптимізовані для збереження та обробки великих обсягів даних у будь-який час, одночасно подаючи їх і виводячи через Інтернет-сервери, де користувачі можуть отримати доступ до своїх даних без затримки.

Інструменти сховища даних дозволяють ефективніше керувати даними, оскільки це дозволяє знаходити, отримувати доступ до них, візуалізувати та аналізувати дані для прийняття кращих бізнес-рішень та досягнення більш бажаних бізнес-результатів. Крім того, вони будуються з урахуванням

експоненціального зростання даних. Немає ризику захаращення складів збільшенням обсягу даних, що зберігаються.

Найбільшою перевагою сховищ даних є можливість перетворення необроблених даних в інформацію та розуміння. Сховища даних пропонують ефективний спосіб підтримувати запити, аналітику, звітність, а також надавати прогнози та тенденції на основі зібраних даних. Дизайн та очищення даних повинні підтримуватися правильним сховищем. Зазвичай сховища даних залежать від великих потужностей, які є надійними, мають менші витрати та добре працюють.

Можливо, ви чули про термін "Hadoop", який кидають раз у раз, але досі не знаєте, що це, що добре. Хоча це ціла тема сама по собі, ми коротко її пояснимо. Hadoop - це програмна база, призначена для розподіленого зберігання та обробки великих даних для обробки великих обсягів даних та обчислень. Hadoop робить революцію в аналітиці великих даних для корпоративного зберігання.

Інший спосіб зберігання великих обсягів даних - це хмарне зберігання, з яким люди знайомі більше. Якщо ви коли-небудь використовували iCloud або Google Drive, це означає, що ви використовували хмарне сховище для зберігання своїх документів та файлів. За допомогою хмарного сховища дані та інформація зберігаються в електронному режимі в Інтернеті, де до них можна отримати доступ з будь-якого місця, відмінюючи необхідність прямого підключеного доступу до жорсткого диска або комп’ютера. Завдяки такому підходу ви можете зберігати практично безмежну кількість даних в Інтернеті та отримувати доступ до них де завгодно.

Хмара забезпечує не тільки доступну інфраструктуру, але й можливість швидко масштабувати цю інфраструктуру для управління значним збільшенням трафіку або використання.

Хмара також забезпечує легку доступність та зручність використання. Коли ви хочете отримати доступ до своїх даних у хмарі, все, що вам потрібно зробити, це ввести свої облікові дані, і ви отримаєте доступ. Все, що вам

потрібно, це підключення до Інтернету та пристрій для доступу до хмари, такий як мобільний телефон або портативний комп’ютер. Хмарне сховище значно покращило продуктивність та ефективність бізнесу, оскільки співробітники можуть миттєво обмінюватися файлами, отримувати доступ до них та редагувати їх.

На додаток до попередніх переваг, хмарне зберігання також значно дешевше, ніж фізичне зберігання даних. Сховища даних споживають велику кількість енергії, простору, ресурсів і мають більший ризик. Однак завдяки хмарному сховищу істотна економія коштів.

1.6. Основні властивості та переваги платформ великих даних

До основних властивостей платформ великих даних можна віднести:

- забезпечення ефективного зберігання та обробки даних, а також їх інтеграції, управління, витягнення, трансформації, та завантаження (ETL);
- використання системи Hadoop: забезпечують функції для масового зберігання даних будь-якого типу, величезну потужність обробки та можливість обробляти практично необмежену кількість паралельних задач;
- потокові обчислення: забезпечують функції для затягування даних у потік, обробки даних та передачі їх назад єдиним потоком;
- функції розвиненої аналітики та машинного навчання;
- функції управління життєвим циклом контенту та документів;
- функції інтеграції великих даних з будь-якого джерела;
- управління даними: містять комплексну систему безпеки, рішення для управління даними та забезпечують дотримання вимог щодо захисту даних.

До головних переваг платформи великих даних та ПЗ аналітики великих даних можна віднести:

- точні дані.

Платформа великих даних пропонує точні дані, що сприяє прийняттю правильних рішень. Її аналітичні засоби зменшують ризик отримання недостовірних даних, які виникають внаслідок використання сирих, не проаналізованих даних;

- підвищення ефективності праці.

Платформа спрощує отримання джерела необхідної інформації. Пропонує також інформацію, що може стати у нагоді в майбутньому, таким чином, зберігаючи час та підвищуючи ефективність роботи користувачів;

- швидкі відповіді на складні питання.

Ефективне управління бізнесом вимагає швидких адекватних відповідей на критичні питання, які впливають на успішність бізнес-операції. Платформа великих даних дозволяє робити це більш надійно. Деякі критичні питання, відповіді на які вимагають тижнів або місяців, за наявності правильного інструменту можуть вирішуватись лише за кілька годин або хвилин;

- безпека даних.

Забезпечує безпечну інфраструктуру, яка гарантує безпеку даних.

Задачі та ПЗ великих даних

Програмні засоби великих даних можна класифікувати за задачами, які вони вирішують. У процес створення рішення повинні бути інтегровані різні засоби для зберігання, управління та аналізу великих даних. Інструменти великих даних відповідно до їх задач включають наступні групи:

- ПЗ зберігання великих даних;
- ПЗ управління великими даними;
- ПЗ обробки великих даних;
- методи та засоби візуалізації великих даних;
- методи та засоби аналітики великих даних. Таким чином, відповідний фреймворк повинен відображати інструменти для зберігання, управління та обробки великих даних, інструменти та методи аналітики, візуалізації та оцінювання у різні етапи процесу побудови рішення. Аналітика великих даних може застосовуватись до виявлення знань та обгрунтованого прийняття рішень.

Задачі та ПЗ великих даних

Програмні засоби великих даних можна класифікувати за задачами, які вони вирішують. У процес створення рішення повинні бути інтегровані різні засоби для зберігання, управління та аналізу великих даних. Інструменти великих даних відповідно до їх задач включають наступні групи:

- ПЗ зберігання великих даних;

- ПЗ управління великими даними;
- ПЗ обробки великих даних;
- методи та засоби візуалізації великих даних;
- методи та засоби аналітики великих даних.

Таким чином, відповідний фреймворк повинен відображати інструменти для зберігання, управління та обробки великих даних, інструменти та методи аналітики, візуалізації та оцінювання у різні етапи процесу побудови рішення. Аналітика великих даних може застосовуватись до виявлення знань та обгрунтованого прийняття рішень.

Зберігання та управління великими даними

Традиційні методи структурованого зберігання та витягування даних, такі як реляційні БД, вітрини або SQL сховища даних, мають певні обмеження, які роблять їх не придатними для роботи з великими даними, а саме вони:

- не дозволяють включати нові джерела даних без їх попереднього очищення та інтеграції,
- не дозволяють швидко виробляти та адаптувати дані,
- не забезпечують можливості синхронізації логічного та фізичного вмісту БД з швидкою еволюцією даних,
- не забезпечують поточні потреби аналізу даних.

Необхідність порівняно недорогого зберігання та обробки гігантських обсягів неструктурованої інформації призвела до створення спеціалізованого ПЗ, яке дозволило розподіляти дані за кластерами з сотень та тисяч вузлів, а також обробляти їх у паралельному режимі. Засоби нового покоління для зберігання та управління не структурованими даними, а саме NoSQL БД, дозволили використовувати репозиторій даних без додаткових розробок, підготовки або налагодження, забезпечили високу масштабованість, розподілення даних між вузлами та швидкий доступ до даних, що зберігаються в оперативній пам’яті. NoSQL [2] БД дозволяють записувати

задачі управління даними на прикладному рівні. Кожна база, в даному випадку, є колекцією незалежних документів, де кожний документ підтримує власні дані та схеми та може мати метадані – оглядову інформацію про дані документа. Прикладна програма може мати доступ до багатьох БД, розташованих у різних місцях. Нові вимоги до зберігання, управління та обробки даних обумовили виникнення Hadoop – фреймворка з відкритим кодом під крилом Apache Sotware Foundation, що дозволяє створювати розподілені системи на базі відносно недорогого обладнання масового попиту. З часом Hadoop був розширений набором бібліотек та утиліт, та сформував навколо себе екосистему проектів з розподіленої обробки даних. Розглянемо його більш детально.

Висновки

У даному розділі було наведено визначення Великих даних, розглянуто основні характеристики Великих даних.

Подано поняття платформи Великих даних та її основних властивостей.

РОЗДІЛ 2

ДОСЛІДЖЕННЯ ТА АНАЛІЗ МОЖЛИВИХ РІШЕНЬ NOSQL

2.1. NoSQL - Рішення для проблем зберігання великих даних

Ідея розробки реляційних баз даних полягала в забезпеченні підходу до зберігання даних, який використовує мову структурованих запитів або SQL [4]. Введення цих баз даних датується 1970-ми роками, коли дані схеми були не такими складними, як сьогодні. Більше того, зберігання було дорогим, і не всі дані розглядалися для архівних. З ростом платформ соціальних медіа обсяги даних, що зберігаються про події, об'єкти та людей зросли в геометричній прогресії. Використання даних у цей час і вік не обмежується лише архівуванням даних, але це також поширюється на частий пошук та обробку даних, щоб служити таким цілям, як генерація в режимі реального часу канали [5] та рекламні оголошення[6], окрім багатьох інших.

Внаслідок складності інформації, що обробляється, і необхідності лікування кількох, порядку сотень, запитів бази даних лише для того, щоб відповісти на один запит API або відтворити веб-сторінку, вимоги сучасної системи бази даних постійно зростають. Одними з ключових факторів у цьому домені є потреба в інтерактивності, складність якої зростає та постійно розвиваються мережі користувачів [7]. Для того, щоб задовольнити ці зростаючі вимоги, застосовуються стратегії розгортання та вдосконалена обчислювальна інфраструктура. З урахуванням цього, серверні розгортання є дорогими і дуже складними, що спричинило зсув до використання хмарного обладнання. Крім того, використання гнучких методів також скоротило час розробки та розгортання, що дозволяє швидше реагувати на потреби користувачів.

Не буде неправильним стверджувати, що реляційні бази даних не створювались для управління спритністю та масштабованістю вимоги сучасних систем.

Недоліки можна усунути за допомогою двох основних технічних підходів, які були розглянуті нижче:

Ручне шардування

Для того, щоб використовувати розподілену парадигму, таблиці потрібно сегментувати на менші одиниці, які потім необхідно зберігати на різних машинах. Цей процес розщеплення називається ручним шардуванням [8].

Однак ця функціональність недоступна в традиційній базі даних, і її має впроваджувати розробник. Більше того, зберігання даних у кожному екземплярі виконується в анонімному режимі. Код програми несе відповідальність за сегментацію даних, зберігання їх розподіленим способом та виконання управління запитами та сукупні результати, які будуть представлені користувачеві. Вимагається додатковий код для підтримки перебалансування даних, виконання операцій об’єднання, обробки відмов ресурсів та реплікації. Ручне шардування може зменшити деякі переваги реляційних баз даних, таких як цілісність транзакцій.

Розподілений кеш

Кешування [9] - це загальновживаний процес, який в основному застосовується для поліпшення читання продуктивності системи. Примітно, що використання кеш-пам'яті не впливає на продуктивність запису і здатний істотно збільшити складність загальної системи. Тому, якщо вимоги системи вимагають інтенсивного читання, тоді слід враховувати використання розподіленого кешу. З іншого боку, увімкнено програми, що вимагають інтенсивного запису чи читання / запису та не потребують розподіленого кешу [10].

Бази даних NoSQL, як відомо, пом'якшують проблеми, пов'язані з традиційними базами даних. До того ж вони також розкривають справжню

потужність хмари, використовуючи товарне обладнання, що зменшує вартість та спрощує розгортання, що значно полегшує життя розробника, оскільки немає необхідності підтримувати кілька шарів кешу більше.

Деякі переваги рішень NoSQL перед традиційними базами даних такі:

1. Масштабованість

NoSQL дозволяє системам масштабуватися горизонтально. Більше того, це можна зробити швидко без впливу на загальну продуктивність системи за допомогою хмарних технологій. Традиційне масштабування бази даних вимагають ручного шардування, що передбачає великі витрати та складність. З іншого боку, NoSQL рішення пропонують автоматичне шардування, зменшуючи складність, а також вартість системи [11].

2. Продуктивність

Як вже згадувалося раніше, системи NoSQL можна масштабувати за необхідності. Зі збільшенням кількості систем, продуктивність системи також відповідно покращується. Справа в тому, що ці системи передбачає автоматичне підбиття значень означає, що накладні витрати, пов'язані з цим, також усуваються, що додатково сприяє підвищенню продуктивності системи.

3. Висока та глобальна доступність

Реляційні бази даних залежать від первинних і вторинних вузлів для виконання вимог доступності. Це не тільки додає складності системи, але й робить систему помірно доступною. Навпаки, рішення NoSQL використовують архітектуру без майстра, а дані розподіляються по декількох системах. Тому навіть після відмови вузла доступність програми залишається доступною не впливає як на операції читання, так і на запис.

Рішення NoSQL пропонує реплікацію даних на всі ресурси. Отже, взаємодія з користувачами є послідовною незалежно від місцезнаходження користувача. Більше того, це також відіграє значну роль у зменшенні латентності з додатковою перевагою переключення уваги розробника з адміністрування баз даних на бізнес.

4. Гнучке моделювання даних

Можливо реалізувати гнучкі моделі даних у NoSQL [11]. Це дозволяє розробникам реалізувати параметри запитів та типи даних, які підходять додатку, замість тих, які відповідають схемі. В процесі взаємодія між базою даних та додатком спрощується, що робить цей підхід кращим варіантом спритного розвитку.

NoSQL - загальний термін, що використовується для опису безлічі технологій, котрі передбачають наступні загальні характеристики [12].

Динамічні схеми

Реляційним базам даних властива вимога створювати схеми заздалегідь. Дані додаються до баз даних лише після того, як ця вимога буде виконана. Наприклад, якщо системі потрібно зберігати дані працівника, такі як ім'я, вік, стать та заробітну плату, тоді таблиця, створена для них, повинна мати відповідну схему.

Така вимога непридатна для гнучких середовищ розробки, оскільки поля даних, можливо, доведеться змінити через деякий час. Можливо, буде додано нову вимогу, як частину ітерації, і згодом, можливо, доведеться мати схему змінений. Якщо база даних велика, це трудомістке завдання. Як результат, базу даних, можливо, доведеться закрити для будь-якого використовування протягом тривалого часу для внесення необхідних змін. Більше того, якщо процес розробки вимагає кілька ітерацій, базу даних, можливо, доведеться закривати досить часто на значний проміжок часу. Очевидно, реляційні бази даних непридатні для зберігання великих, неструктурованих та невідомих даних.

NoSQL задовольняє цю вимогу, оскільки не має заздалегідь визначених схем. Більше того, вставка даних не вимагає розробника визначити схему заздалегідь. Як результат, зміни в структурі даних можуть бути внесені в режимі реального часу без необхідності закривати базу даних для будь-якого іншого використання [12]. Є кілька переваг використання цього підходу.

Окрім того, що це зменшує час адміністратора, такий підхід також скорочує час, необхідний для розробки та спрощує процес інтеграції коду.

Автозаточування

Реляційні бази даних побудовані таким чином, що вони повинні мати сервер, який контролює решту системи для забезпечення вимог надійності та доступності рішення бази даних. Тому така система може тільки підтримка вертикального масштабування, що не просто дорого, але призводить до створення невеликої кількості точок відмови. Крім цього, це також обмежує масштаб, який може підтримувати система. З огляду на системні вимоги, рішення бази даних має підтримувати горизонтальне масштабування. Тому він повинен додавати сервери до ансамблю та позбутися обмежень, які зосереджені на тестуванні ємності одного сервера. Хмарні обчислення пропонують найкраще рішення в цьому відношенні, надаючи послуги на замовлення та необмежену масштабовану здатність. Отже, системі більше не потрібно покладатися на один сервер для задоволення своїх потреб. Ще одна важлива грань використання хмари - це вбудоване адміністрування баз даних. Більше того, розробнику більше не потрібно створювати складні платформи і можуть просто зосередитись на написанні коду програми. Нарешті, використання хмарних, багаторазових серверів коштують значно менше, ніж одиничний сервер великої ємності.

Для того, щоб виконати шардінг бази даних, що охоплює кілька серверів, потрібно зробити складні механізми кілька серверів діють єдиною системою, їх потрібно встановити. З іншого боку, бази даних NoSQL підтримують автоматичне затухання. Іншими словами, база даних автоматично розподіляє дані між кількома системами без необхідності використання адміністратора, щоб бути в курсі складу пулу серверів. Балансування навантаження [13] для даних та запитів також є автоматично виконується системою. Це дозволяє системі запропонувати високу доступність. Як і коли сервер знижується, його можна зручно замінити, а операції залишаються незмінними.

Автоматична реплікація

Реплікація виконується автоматично для будь-якої системи NoSQL. Таким чином, система може відновити стихійні лиха досить легко, також дозволяючи високий рівень доступності. З точки зору розробника, потрібно враховувати ці аспекти розвитку в коді програми.

Інтегрований кеш

Інтегровані можливості кешування систем NoSQL досить добре оснащені і більшість з них часто використовуються дані зберігаються в системній пам'яті для забезпечення швидкого доступу. Тому немає необхідності підтримувати багаторазове кешування шари на рівні програми.

2.2. Моделі зберігання великих даних

NoSQL - це технологія, яка розроблена для протидії проблемам, представленим реляційними базами даних, що є реалізовані різними способами за різними моделями. Загальні характеристики моделей NoSQL включають ефективність зберігання, зниження експлуатаційних витрат, висока доступність, висока паралельність, мінімальне управління, висока масштабованість і низька затримка [14]. Рішення NoSQL класифікували за кількома критеріями. Yen [15] надав детальну класифікацію рішень NoSQL, розділивши їх на дев'ять категорій, які включають широкий стовпчастий магазин, сховище документів, база даних об’єктів, сховище кортежів, сервер структур даних, сховище ключових значень, кеш-ключ-значення, впорядковане сховище ключових значень і з часом схоже сховище ключових значень. Ще одне таксономічне дослідження було проведено Нортом [15], який дав вичерпну класифікацію та включав хмарні рішення, також для аналізу. Рішення були класифіковані за шістьма категоріями, а саме "Дані про сутність-атрибут-вартість" Магазини, магазини стовпців Amazon

Platform, сховища даних Key-Value та сховища документів розподіленої хеш-таблиці.

Catel [16.] та Leavitt [17] запропонували класифікацію на основі моделі даних. Catel [16] розділяє рішення NoSQL на три категорії, а саме магазини ключових значень, магазини документів та розширювані магазини записів. З іншої сторони, Leavitt [17] пропонує використовувати три категорії, а саме на основі документів, сховища ключових значень, сховища, орієнтовані на стовпці.

Скофілд [18] дав найбільш прийняту схему категоризації, класифікуючи бази даних на реляційні, графічні, сховища значень документа, стовпця та ключа. Ця дослідницька робота використовує вищезазначену основу для класифікації та охоплює сховища, орієнтовані на документи, модель даних графіків, сховище ключових значень та широке сховище стовпців у наступних розділах.

2.3. Модель даних, орієнтована на документ

База даних NoSQL, яка використовує документи для зберігання та пошуку даних, називається документоорієнтованою базою даних [19]. Інші назви цієї моделі даних NoSQL - це база даних документів та сховище документів. Це насамперед використовується для управління напівструктурованими даними через його гнучкість та підтримку змінної схеми. Згадані раніше, бази даних використовують документи для своєї роботи. Ці документи можуть бути у форматі PDF або word форматі. Однак блоки JSON та XML є найбільш поширеними форматами документів. Реляційна база даних містить стовпці, які описуються їх іменами та типами даних. Навпаки, у випадку з документом бази даних, опис типу даних та значення відповідного опису наводяться в документі.

Структура різних документів, що складають базу даних, може бути подібною або різною за структурою. Очевидно, немає потреби змінити схему додавання даних до бази даних.

Документи групуються разом, утворюючи структуру, яка називається колекцією. Може бути кілька колекцій баз даних. Ця структура подібна до функціонування таблиці, яка присутня в реляційній базі даних.

Бази даних, орієнтовані на документи, забезпечують механізм виконання запитів до колекцій та отримання документів, які задовольнити вимоги до атрибутів. Існує кілька переваг використання цього підходу, які включають:

Більша частина зростаючих даних надходить з пристроїв IoT, соціальних медіа та Інтернету. Однак ці дані не вписуються у стандартні моделі даних додатків. Орієнтовані на документи бази даних пропонують гнучке моделювання даних, на відміну від реляційних баз даних, які змушують програми вбудовувати дані в існуючі моделі незалежно від їх потреби.

Ефективність записування баз даних, орієнтованих на документи, краща за звичайні системи [20]. Щоб зробити систему доступною для написання, база даних може також порушити консистенцію даних. Отже, навіть якщо система виходить з ладу і реплікація займає більше часу, ніж очікувалося, операція запису буде швидкою.

Відомо, що функції індексації та механізми запитів баз даних, доступні в цій категорії, є швидкими та ефективними. Тому вони пропонують більш швидку продуктивність запитів.

Кожен документ може зберігати в собі, як правило, XML, JSON або BSON - бінарному-зберігається JSON. Але зараз це вже практично завжди JSON або BSON. Це теж як би пара ключ-значення, можна це собі уявити як таблицю, в якій кожен рядок має певні характеристики, і ми за цими ключам можемо з неї що-небудь діставати.

Перевага документоорієнтованих БД: вони мають дуже високою доступністю і гнучкістю даних. У будь-який документ, в будь-який JSON можна записати абсолютно будь-який набір даних. І вони дуже часто застосовуються - наприклад, коли потрібно зробити який-небудь каталог і коли кожен продукт в каталозі може мати різні характеристики.

Або, наприклад, профілі користувачів. Хтось вказав свій улюблений фільм, хтось - улюблену їжу. Щоб не засовувати все в одне поле, яке буде зберігати незрозуміло що, ми можемо все записати в JSON документоорієнтованих бази.

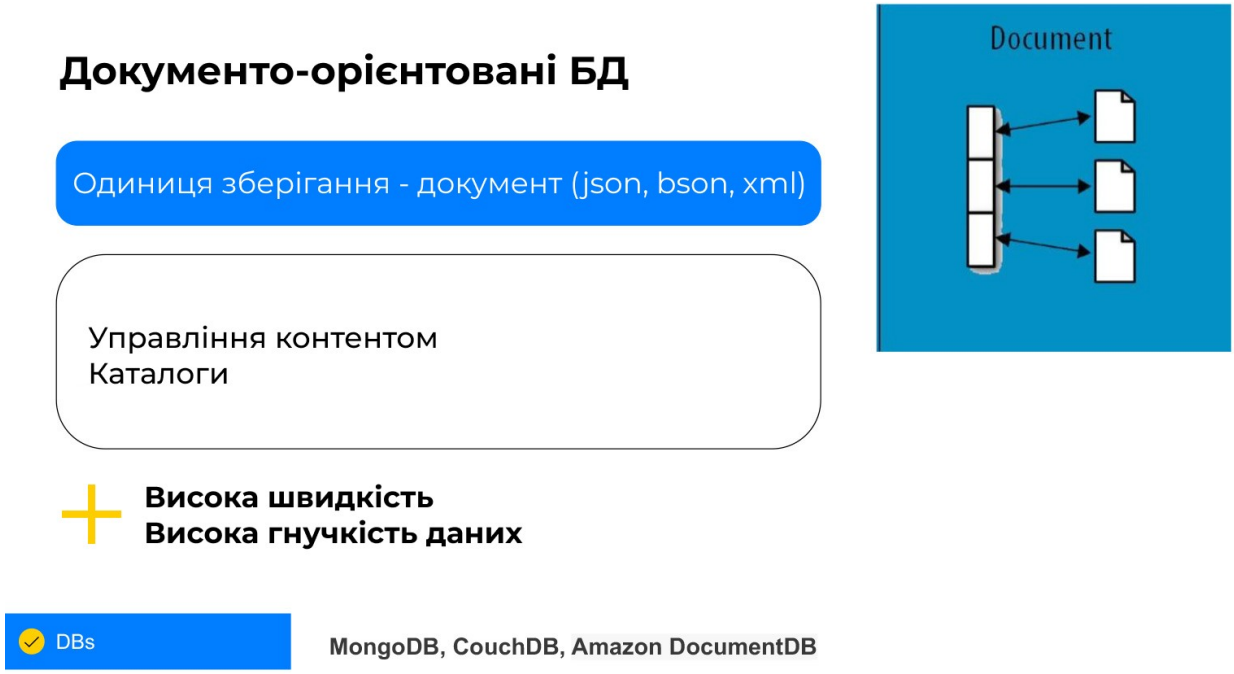

Рис. 2.1 – Опис документо-орієнтованої бази даних

2.4. Графова модель бази даних

Ця модель даних NoSQL створена спеціально для підтримки зберігання та обробки об'ємних даних, які можуть бути напівструктурованими, структурованими або неструктурованими за типом. Таким чином, дані можуть бути доступні та отримані з різних джерел. Як результат, модель даних Graph широко використовується в соціальних мережах та аналітиці

великих даних. Заслуговує на увагу те, що реляційні бази даних були розроблені для зберігання структурованої інформації, доступної та генерованої для підприємств. Отже, схема даних, які слід зберігати, доступна заздалегідь. Навпаки, дані, генеровані IoT (Інтернет речей) та соціальні медіа неструктуровані. Більше того, він генерується в режимі реального часу.

Графові бази даних є хорошим варіантом для зберігання неструктурованих даних, що генеруються такими різноманітними джерелами швидкості. Немає необхідності визначати схему перед зберіганням даних, що робить базу даних досить гнучкою. До того ж бази даних графів є економічно вигідними та динамічними, коли йдеться про інтеграцію даних, що надходять з різних джерел. Більше того, графові бази даних краще обладнані для обробки, зберігання та обробки високошвидкісних даних як порівняно з реляційними базами даних.

Вищезгадані програми, такі як аналіз соціальних медіа та аналітичні рішення на основі IoT, потребують базової технології для інтеграції даних, що надходять з неоднорідних джерел, та встановлення зв’язків між різними наборами даних. Дані програми такого роду найкраще обробляти за допомогою бази даних семантичних графів або RDF triplestore [21], що фокусується на взаємозв'язку між різними елементами бази даних та генерує аналітику на цій основі. Графові бази даних в основному використовуються для аналізу в режимі реального часу через їх здатність обробляти великі обсяги даних та без необхідності заздалегідь визначати схему.

Переваги використання графових баз даних можна узагальнити наступним чином:

1. Інтеграція вхідних даних з різних джерел обмежена, коли схему потрібно визначити раніше додавання даних, оскільки додавання нового джерела може зажадати зміни схеми, що вимагає як часу, так і складності. У базах даних, де такої потреби немає, інтеграція даних безмежна, проста і економічно вигідна [22].

2. Графові бази даних пропонують додаткову підтримку онтологій або семантично багатих схем даних [22]. Отже, організації можуть створювати логічні моделі будь-яким чином, як їм заманеться.

3. Графові бази даних використовують міжнародні стандарти для представлення даних в Інтернеті [22]. Це результати для полегшення інтеграції та обміну даними. Уніфікований ідентифікатор ресурсу (URI) є одним із стандартів і використовується для представлення даних у базах даних семантичних графів. URI - це унікальний ідентифікатор, який використовується для розрізнення між пов’язаними об’єктами. Наявність такого чіткого підходу до ідентифікації суб'єктів робить доступ і пошук простіший, що робить підхід економічно вигідним. Більше того, це також полегшує обмін даними, стосується відображення даних у зв’язаних (відкритих) даних.

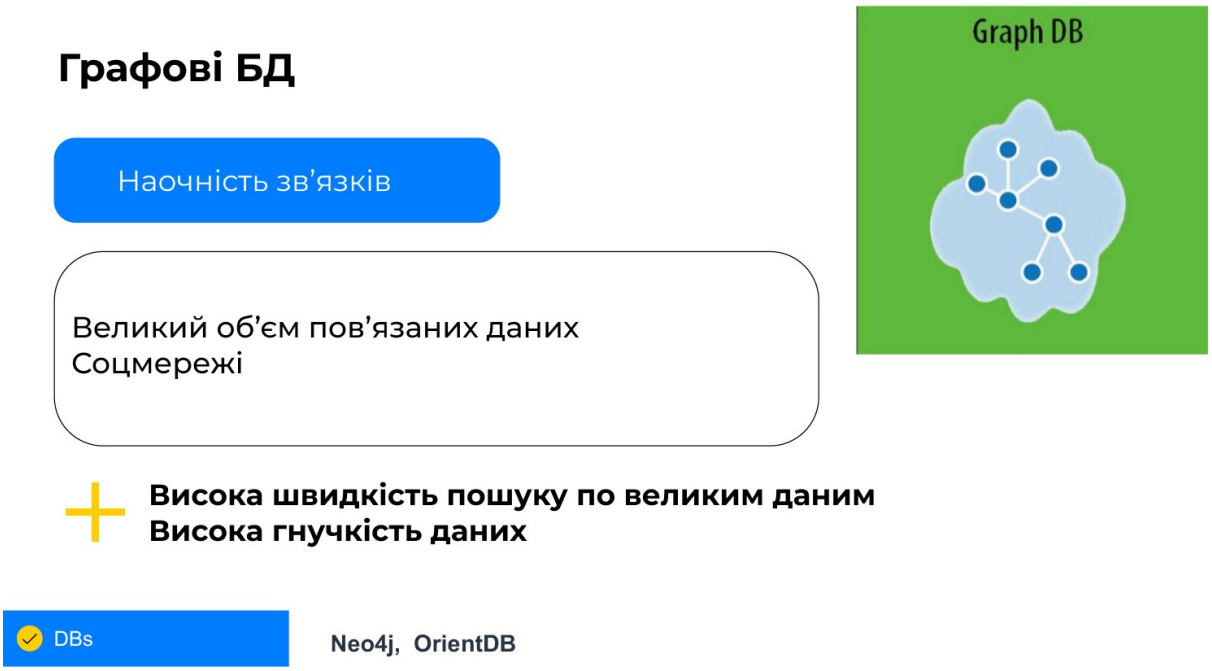

Рис. 2.2 – Опис графової бази даних

2.5. Модель даних ключ-значення

Найбільш гнучким типом бази даних NoSQL є ключ-значення [23], який реалізує політику без схем не має схеми і робить значення даних непрозорими. Значення даних може зберігати рядки, числа, зображення, двійкові файли, лічильники, XML, JSON, HTML та відео, крім багатьох інших [23]. Доступ до збережених значень можна отримати за допомогою ключа. Гнучкість бази даних проявляється в тому, що програма контролює значення даних, повністю. Основні переваги моделі ключ-значення такі:

1. База даних не змушує програму структурувати свої дані у певній формі. Тому додаток може безкоштовно моделювати свої дані відповідно до вимог випадку використання.

2. До об’єктів можна просто отримати доступ за допомогою ключа, присвоєного об’єкту. Використовуючи цю базу даних, немає необхідності виконувати такі операції, як об'єднання, об'єднання та блокування об'єктів [23], які роблять ці дані модель, найефективніша та найефективніша.

3. Більшість доступних баз даних ключ-значення дозволяють змінювати масштаби, коли виникає попит на них. Більше того, це можна зробити за допомогою товарного обладнання без необхідності будь-якого перепроектування.

4. Забезпечити високу доступність набагато простіше і нескладніше із сховищами ключових цінностей. Розподілений архітектура та безконфігураційна конфігурація деяких доступних баз даних цього типу забезпечує вищу стійкість [23].

5. Дизайн цих баз даних такий, що легко додавати та зменшувати потужність. Більше того, ці бази даних краще обладнані для усунення несправностей мережі та апаратних несправностей [23].

Ми можемо використовувати ключ-значення, наприклад, для зберігання сесій користувача. Користувач клікнув, ми записали це в value. Це такий schemaless, модель даних без певної схеми, структури значень. Завдяки тому, що це дуже проста структура, вона має високу швидкість і легко

масштабується. У нас вже є ключі, і ми можемо дуже легко їх шардіровать, зробити їх хеші. Це одна з найбільш високомасштабіруемих баз даних.

Приклади - Redis, Memcached, Amazon DynamoDB, Riak, LevelDB. Ви можете подивитися особливості реалізації саме key-value сховищ.

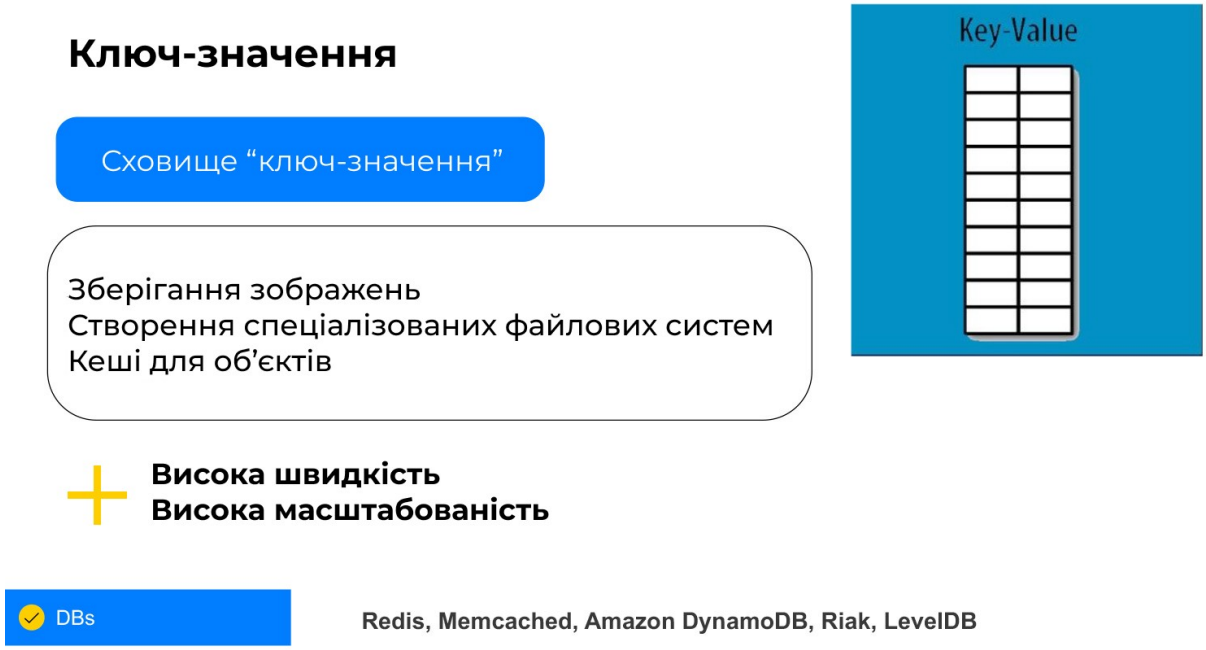

Рис. 2.3. Опис бази даних ключ-значення

2.6. Стовпчикова модель даних

Магазини широких стовпців [24] мають стовпці та сімейства стовпців як базові сутності. Факти або дані групуються разом для формування стовпців, які впорядковані у вигляді сімейств стовпців, які є конструкціями, подібними до таблиць у реляційні бази даних. Наприклад, дані про особу, такі як ім'я, ім'я та адреса, є фактами про окремі та можуть бути згруповані разом, щоб утворити рядок у реляційній базі даних. Навпаки, такі самі факти організовані у вигляді стовпців у магазині з широкими колонками, і кожна з колон включає кілька груп.

Отже, один широкий стовпець може зберігати дані, еквівалентні тим самим, що зберігаються багатьма рядками в реляційній базі даних.

Інші назви таких баз даних включають орієнтовану на стовпці СУБД, стовпчасті бази даних та сімейства стовпців.

Ключові переваги [24] використання баз даних з широким стовпцем:

1. Розбиття та стиснення даних можна ефективно виконувати, використовуючи широкі бази даних зберігання стовпців.

2. Запити AVG, SUM і COUNT, можуть виконуватися ефективно і завдяки цьому властива цій базі даних ефективна структура.

3. Цей тип бази даних є дуже масштабованим і добре підходить для систем масової паралельної обробки (MPP).

4. Таблиці з величезними обсягами даних можна завантажувати та запитувати майже за короткий час, що робить час відгуку бази даних відносно низький.

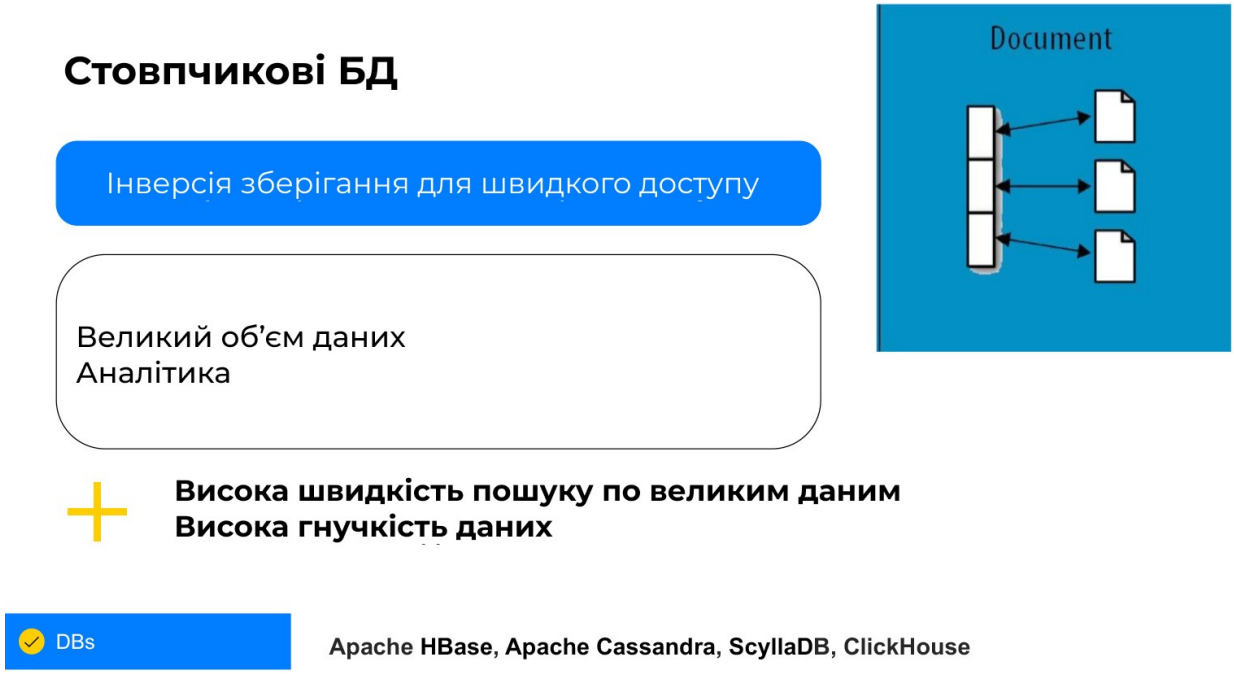

Рис. 2.4. Опис стовпчикової бази даних

Уявімо, що у нас є величезна таблиця. І якби ми зберігали дані рядками, то було б те, що внизу: величезна кількість рядків. Для відібрання навіть за трьома параметрами цієї таблиці нам потрібно пройти по всій таблиці. А коли ми зберігаємо значення за стовпцями, то при відборі за трьома значеннями потрібно пройтися тільки за трьома ось таким, грубо

кажучи, рядках, тому що стовпці у нас записуються ось так. Проходячи по цих трьох рядках, ми відразу отримуємо порядковий номер потрібного нам значення і дістаємо його вже з інших стовпців.

У чому перевага таких баз даних? За рахунок того, що вони шукають по маленькому обсягом даних, у них дуже висока швидкість обробки запитів і велика гнучкість даних, тому що ми можемо додавати будь-яку кількість стовпців, не змінюючи структуру. Тут не як в реляційних БД, нам не потрібно засовувати наші дані в певні рамки.

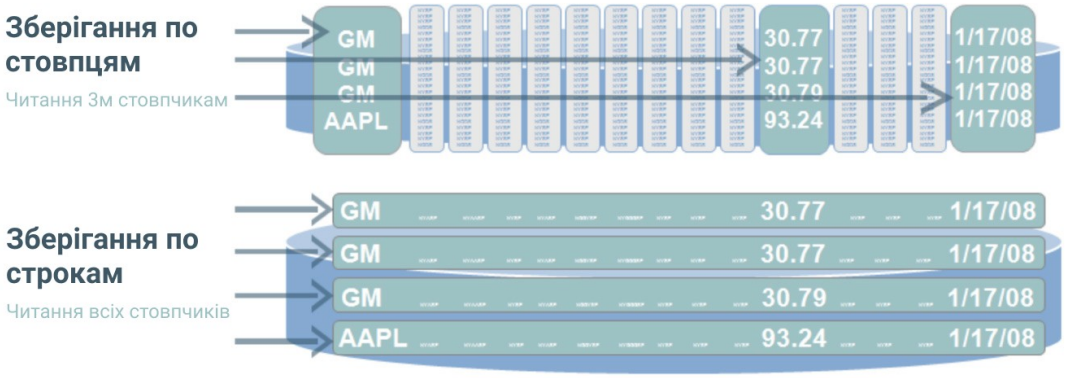

Рис. 2.5. Стовпчикова БД

Найпопулярніші стовпчикові бази даних - це, напевно, Cassandra, HBase і ClickHouse. Випробуйте їх. Дуже цікаво інвертувати в голові ставлення рядків і стовпців. І це дійсно ефективний і швидкий доступ до великої кількості даних.

2.7. Вибір рішення NoSQL для системи великих даних

Одне з головних технологічних рішень, яке потрібно прийняти під час проектування програми для великих даних, включає вибір рішення NoSQL. Для цього потрібно мати на увазі дві речі. Правильна модель даних для програми залежить від типу даних, з якими потрібно обробляти. Класифікація баз даних NoSQL на основі їх моделей даних наведена в попередньому розділі. На успіх програми великих даних може сильно

вплинути невідповідність моделі даних програми та обраного рішення NoSQL. Ще одна важлива при виборі рішення NoSQL враховується вимога до масштабованості. Дуже важливо зрозуміти, що деякі рішення NoSQL можуть масштабуватися, як Cassandra, тоді як інші можуть базуватися на пам'яті і не масштабуватися на різних машинах.

Як уже згадувалося раніше, правильна модель даних для програми залежить від даних, з якими вона має працювати. Наприклад, якщо дані програми можуть бути представлені у вигляді графіка, то модель графіка є найбільшою відповідна модель даних для програми. Кожна модель даних найкраще підходить для певного набору програм і вимоги. У цьому розділі розглядаються критерії вибору для вирішення, яка модель великих даних підходить для конкретного випадку дослідження з огляду на його модель домену, схеми доступу до даних та випадки використання.

Документоорієнтовані бази даних обходять документи. Отже, документ є основною атомною одиницею зберігання в такій базі даних. Будь-яка модель домену, яка дозволяє розділяти та розподіляти свої дані між документами, може використовувати документ база даних. Деякі загальні приклади включають CMS, програмне забезпечення для блогів та wiki-програмне забезпечення. Однак, розглядаючи цю модель даних, ви можете зустріти випадки використання, коли реляційна модель може бути настільки ж хорошим варіантом використовувати як нереляційну базу даних.

Зберігання ключових значень - це загальновживані моделі даних, переважно для областей додатків, що оточують дані, наприклад, користувача профіль, електронні листи, коментарі до блогу / статті, інформація про сеанс, дані кошика покупок, огляди товарів, деталі товару та На додаток до багатьох інших таблиці пересилання протоколів Інтернету (IP). Дуже важливо зрозуміти, що ключ-значення store можна використовувати для зберігання повних веб-сторінок [25]. У цьому випадку URL-адресу можна використовувати як ключ, а вміст веб-сторінки, як значення. Однак інші

моделі даних можуть бути більш підходящими для цієї мети, якщо програма цього вимагає.

Графові бази даних пропонують ефективний спосіб управління даними та їх поєднання. Дані підприємства, як правило, пов'язані між собою графові бази даних для їх зберігання забезпечують більш просте управління вмістом. Більше того, персоналізація також досягається простіше. На додаток до цього, концепція пов'язаного світу, темпи якої особливо піднялися після зростання соціальних медіа та IoT, можуть скористатися тим фактом, що графові бази даних дозволяють інтегрувати неоднорідні, взаємопов’язані дані з різних джерел.

Магазини з широкими колонками утворюють останню категорію моделей великих даних. Ці бази даних вважаються найбільш доречними для розподілені системи [26]. Іншими словами, якщо доступні дані великі і їх можна розподілити між машинами, то база даних магазину з широким стовпцем може бути надзвичайно корисною. Деякі основні переваги використання цієї бази даних - зменшується час запиту для деяких запитів. Однак цей момент повинен бути чітко досліджений до прийняття рішення на користь такого. Для деяких запитів час може бути однаковим або вищим, ніж пропонований звичайною СУБД. Варіанти використання чотирьох моделей великих даних наведені нижче (рисунок 2.6).

Модель великих даних	Кращі для використання випадки	Не кращі для використання випадки
Документоорієнтована	1. Системи управління контентом 2. Платформи електронної комерції 3. Блог-платформи 4. Платформи аналітики	1. Додатки, що вимагають складних пошукових запитів. 2. Додатки, що вимагають складних транзакцій з декількома операціями.
Ключ-значення	1. Зберігання уподобань користувача 2. Обслуговування профілів користувачів, які не мають певної схеми 3. Зберігання даних сеансів для користувачів 4. Зберігання даних кошиків для кількох користувачів	У сценаріях, де: 1. Потрібно запитати конкретне значення даних. 2. Потрібно опрацювати кілька унікальних ключів. 3. Часте оновлення частини вартості. 4. Значення даних встановили взаємозв'язок з один одного, і програма вимагає експлуатації того самого.
Графова	1. Мережеві та IT-операції 2. Пошуки на основі графіків 3. Соціальні мережі 4. Виявлення шахрайства	Така модель недоречна для будь-якого застосування, для якого дані не можуть бути змодельовані як графік.
Стовпчикова	1. Блог-платформи 2. Системи управління контентом 3. Системи на базі лічильників 4. Додатки з інтенсивною обробкою	1. Додаток вимагає складних запитів. 2. Додаток має різну структуру запитів. 3. У сценаріях, де вимога до бази даних не встановлена, використання такого магазину треба уникати

Рис.2.6. Варіанти використання чотирьох моделей великих даних

Обговорюючи придатність рішень NoSQL до реальних проблем, важливо згадати CAP Теорема [27]. Ця теорема вводить поняття толерантності до розділів (P), доступності (A) та узгодженості (C) для розподілених систем і стверджує, що всі ці три характеристики не можуть бути забезпечені рішенням одночасно.

Послідовність - це характеристика, яка гарантує, що всі вузли розподіленої системи повинні читати одне і те ж значення дані у будь-який час. Якщо змінено значення даних, то зміна повинна бути узгодженою для всіх вузлів. Однак, якщо зміна призводить до помилки, тоді необхідно виконати відкат для забезпечення узгодженості.

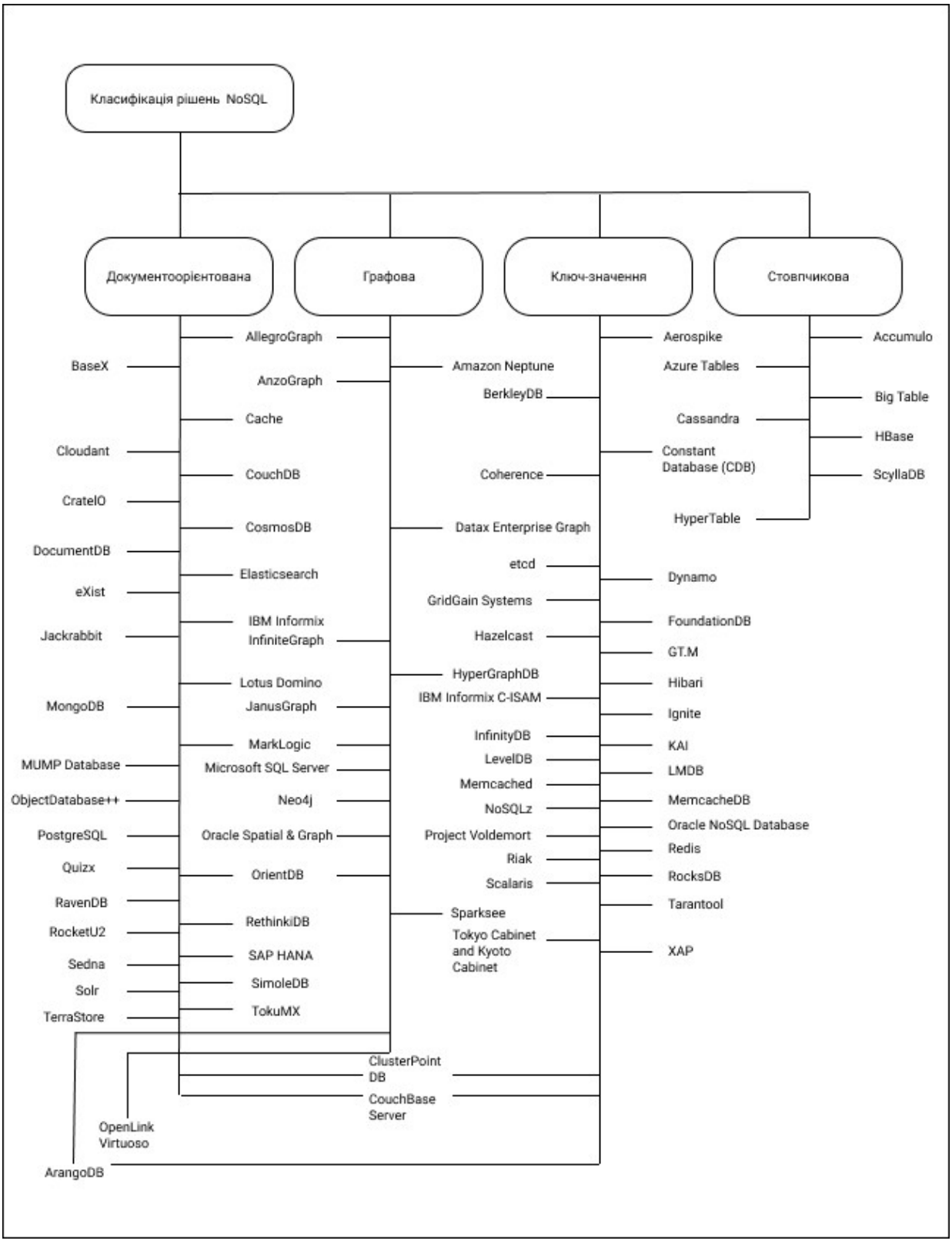

Рис. 2.7. Класифікація рішень NoSQL

Доступність визначає експлуатаційні вимоги системи, які гарантують, що коли надходить запит користувачем, він повинен відповісти на неї, незважаючи на її стан. Толерантність розділів відноситься до здатності

системи працювати, незважаючи на вихід з ладу розділу та втрату повідомлень. Це також можна описати як здатність системи працювати незалежно від збою мережі. Різні рішення баз даних та їхній статус CAP було описано нижче. Зазначено, що жодна розподілена система не може володіти всіма трьома характеристиками. На підставі цього за твердженням, системи NoSQL можуть бути CA (послідовно-доступні), AP (толерантні до доступних розділів) або CP (послідовні толерантні до розділів) [28].

Однією з найбільших проблем при використанні NoSQL є використання відповідної моделі даних. Використання невідповідної моделі може суттєво вплинути на продуктивність системи, що робить це вирішальним технологічним рішенням.

Іншим важливим технологічним рішенням у цьому випадку є вибір правильної моделі розподілу. Масштабування прочитаної операції підтримуються архітектурою ведучого-підлеглого. Однак якщо масштабування як операцій читання, так і запису є бажана, тоді однорангова архітектура - кращий варіант. Використання баз даних NoSQL також має деякі проблеми безпеки що слід розглянути та пом'якшити, перш ніж рішення може бути розроблено та застосовано за допомогою того самого.

Рисунок 2.7 ілюструє класифікацію рішень NoSQL на основі даних. Детальний опис реалізацій разом з підтримуваними моделями даних наведено в таблиці 2.1. «+» вказує на підтримку відповідних даних модель, тоді як «-» не пропонує ніякої підтримки для відповідної моделі даних.

Таблиця 2.1

Аналіз баз даних з вказанням підтримуваних моделей

№	Впровадження	Підтримувана модель даних				Основні характеристики
		Документо-орієнтована	Графова	Ключ-значення	Стовпчикова	
1.	AllegroGraph	+	+	-	-	1. Запатентована 2. Підтримує RDF, JSON та JSON-LD 3. Забезпечує реплікацію Multi-Master 4. Підтримує транзакції ACID, двофазні коміт і повнотекстовий пошук
2.	Accumulo	-	-	-	+	1. Безкоштовно 2. Масштабована та розподілена 3. Він побудований на Hadoop, Thrift та Zookeeper 4. Забезпечує захист та механізми на рівні клітини для програмування на стороні сервера
3.	Aerospike	-	-	+	-	1. Безкоштовно 2. Високо масштабований 3. Оптимізована для флеш-пам'яті, вбудована пам’ять 4. Надійний і послідовний 5. Використовується для таких програм, як динамічний Інтернет портали, профілювання користувачів та виявлення шахрайства
4.	Amazon Neptune	-	+	-	-	1. Запатентована 2. Повністю керована база даних 3. Надається як веб-сервіс 4. Підтримує RDF та моделі графіків властивостей 5. Підтримує SPARQL та TinkerPop Gremlin мови запитів
5.	AnzoGraph	-	+	-	-	1. Запатентована 2. Масово паралельно 3. Графік обробки онлайн-аналітики (GOLAP) база даних 4. Підтримує SPARQL та Cypher 5. Спочатку призначений для аналізу семантичної потрійності дані інтерактивно
6.	ArangoDB	+	+	+	-	1. Безкоштовно 2. Підтримує кілька моделей баз даних за допомогою одноядерний 3. Володіє уніфікованою мовою запитів - Мова запитів ArangoDB (AQL)
7.	Azure Tables	-	-	-	+	1.Запатентована 2.Надається як послуга, де зберігається інформація допущених до колекцій, що можна розділити. Доступ до даних здійснюється за допомогою основних та розділові ключі.
8.	BaseX	+	-	-	-	1. Безкоштовно 2. Забезпечує підтримку JSON, XML та бінарних файлів формати 3. Знаряддя майстра -невольницька архітектура 4. Забезпечує підтримку одночасних структурних та повний

						-оновлення / пошук тексту

Продовження таблиці 2.1

9.	BerkeleyDB	-	-	+	-	1. Безкоштовно (з комерційними версіями) 2. Високопродуктивний та масштабований 3. Підтримує складне управління даними 4. Найбільш підходящий для додатків, що потребують вбудованої бази даних.
10.	BigTable	-	-	-	+	1. Запатентована 2. Висока продуктивність 3. Забезпечує стиснення даних
11.	Cache	+	-	-	-	1. Запатентована 2. Дані зберігаються у багатовимірних масивах. Отже, структуровані дані, що мають ієрархічний характер, можуть зберігатися. 3. Зазвичай використовується для бізнесу та охорони здоров'я відповідні програми
12.	Cassandra	-	-	-	+	1. Безкоштовно 2. Розподілений 3. Високодоступний 4. Майстер-реплікація з надійною підтримкою кластерів у кількох центрах обробки даних
13.	CDB or Constant Database	-	-	+	-	1. Безкоштовна бібліотека 2. Асоціативний масив на диску, який відображає ключі до значень, дозволяючи ключу мати кілька значень 3. Його можна використовувати як спільну бібліотеку.
14.	Cloudant	+	-	-	-	1. Запатентована 2. Поширена служба баз даних 3. Використовує BigCouch та модель JSON у серверній системі
15.	Clusterpoint Database	+	-	+	-	1. Запатентована, але дозволяє завантажувати безкоштовно 2. Розподілена платформа баз даних JSON / XML 3. Транзакції відповідають властивостям ACID 4. Високодоступний 5. Забезпечує заточування та тиражування 6. Використовує SQL або JS як мову запитів
16.	Coherence	-	-	+	-	1. Запатентована 2. Сітка даних в пам'яті та розподілений кеш 3. Підходить для систем, що вимагають високого рівня масштабованість та доступність, зберігаючи затримку на нижчих рівнях
17.	CouchBase Server	+	-	+	-	1. Безкоштовно 2. Розподілена база даних 3. Використовує SQL як мову запитів 4. Використовує модель JSON
18.	CouchDB	+	-	-	-	1. Безкоштовно 2. Підтримує JSON через HTTP / REST 3. Забезпечує обмежену підтримку транзакцій ACID 4. Підтримує багаторівневий контроль паралельності
19.	CrateIO	+	-	-	-	1. Безкоштовно 2. Він базується на екосистемі Elasticsearch / Lucene 3. Підтримує двійкові об'єкти або їх також називають BLOB. 4. Використовує синтаксис SQL для розподіленого запиту системи в режимі реального часу
20.	CosmosDB	+	-	-	-	1. Запатентована

						2. Надано як платформа як послуга 3. На основі DocumentDB

Продовження таблиці 2.1

21.	DataStax Enterprise Graph	-	+	-	-	1. Запатентована 2. Масштабована, розподілена база даних 3. Дозволяє здійснювати запити в режимі реального часу 4. Підтримує Tinkerpop 5. Відомо, що вона добре інтегрується з Cassandra
22.	DocumentDB	+	-	-	-	1. Запатентована 2. Надається як служба баз даних 3. Повністю керована версія MongoDB
23.	Dynamo	-	-	+	-	1. Запатентована 2. Розподілений магазин даних 3. Високодоступний 4. Підтримує поступову масштабованість, симетрію між вузлами, децентралізацію і використовує неоднорідність інфраструктури, на якій працює.
24.	ElasticSearch	+	-	-	-	1. Безкоштовно 2. Підтримує JSON 3. В основному пошукова машина
25.	etcd	-	-	+	-	1. Безкоштовно 2. Підтримує двійкові дані 3. Дозволяє встановлення версій, перевірку, колекції, тригери, кластеризацію, повнотекстовий пошук Lucene, оновлення ACLS та XQuery 4. Використовує XML над REST / HTTP
26.	eXist	+	-	-	-	1. Безкоштовно 2. Підтримує текстові формати JSON, HTML та XML, крім двійкових форматів 3. XQuery - це надана мова запитів, тоді як XSLT – відповідне мова програмування
27.	FoundationDB	-	-	+	-	1. Безкоштовно 2. Відповідає властивостям кислоти 3. Масштабована 4. Дозволяє тиражування 5. Прив'язки для Python, C, PHP та Java, на додаток до багатьох інших мов програмування доступний
28.	GridGain Systems	-	-	+	-	1. Запатентована 2. Надаються послуги та програмні рішення для систем, що працюють з великими даними 3. Підтримує обчислення в пам'яті 4. Забезпечує покращену пропускну здатність та зменшену затримку
29.	GT.M	-	-	+	-	1. Безкоштовно 2. Розроблений для обробки транзакцій 3. Підтримує транзакції ACID 4. Підтримує реплікацію та шифрування бази даних
30.	Hazelcast	-	-	+	-	1. Безкоштовно 2. MapStore може бути визначений користувачем 3. MapStore може бути постійним 4. Висока послідовність та підтримка обміну у формі послідовного хешування
31.	HBase	-	-	-	+	1. Безкоштовно 2. Розподілена база даних 3. Він працює на вершині Hadoop і надає можливості, подібні до можливостей BigTable.

						4. Він відмовостійкий для сценаріїв, де великий обробляється обсяг розріджених даних.
32.	Hibari	-	-	+	-	1. Безкоштовно 2. Розподілене сховище великих даних 3. Високодоступний 4. Сильно послідовний

Продовження таблиці 2.1

33.	HyperGraphDB	-	+	-	-	1. Безкоштовно 2. Схеми динамічні та гнучкі 3. Представлення знань та моделювання даних є ефективними 4. Неблокуючий паралелізм 5. Підходить для випадків використання семантичної мережі та довільних графіків
34.	HyperTable	-	-	-	+	1. Запатентована 2. Заснована на BigTable 3. Масштабована
35.	IBM Informix	+	-	-	-	1. Запатентована 2. СУБД, що підтримує JSON 3. Відповідає правилам ACID 4. Підтримує заточування та тиражування
36.	IBM Informix C-ISAM	-	-	+	-	1. Цей API відповідає відкритим стандартам 2. Дозволяє керувати файлами даних, які були організовані за допомогою індексації B + 3. Це сховище файлів, що використовується Informix.
37.	Ignite	-	-	+	-	1. Безкоштовно 2. Розподілена обчислювальна платформа в пам'яті 3. Забезпечує кешування та платформу обробки 4. Забезпечує підтримку транзакцій ACID та завдань MapReduce 5. Дозволяє розділення, кластеризацію та реплікацію 6. Дуже послідовна
38.	InfiniteGraph	-	+	-	-	1. Запатентована 2. Увімкнено хмару 3. Розподілений 4. Це масштабовано та крос-платформно 5. Він здатний працювати з високою пропускною здатністю
39.	InfinityDB	-	-	+	-	1. Запатентована 2. Повністю розроблений на Java та включає СУБД та механізм баз даних 3. На основі архітектури B-дерева 4. Забезпечує високу продуктивність 3. Знижує ризики, пов'язані з несправностями
40.	Jackrabbit	+	-	-	-	1. Безкоштовно 2. Впровадження сховища вмісту Java
41.	JanusGraph	-	+	-	-	1. Безкоштовно 2. Розподілений 3. Масштабована та добре інтегрується з базовими базами даних, такими як HBase , Cassandra , BigTable та BerkleyDB 4. Добре інтегрується з такими платформами, як Giraph , Spark та Hadoop 5. Забезпечує підтримку повнотекстового пошуку шляхом зовнішньої інтеграції з Solr та Еластичний пошук

42.	KAI	-	-	+	-	1. Безкоштовно 2. Масштабована 3. Висока стійкість до несправностей 4. Забезпечує низьку затримку 3. Використовується для соціальних мереж та веб-сховищ
43.	LevelDB	-	-	+	-	1. Безкоштовно 2. Веде байтові масиви для зберігання пар ключів і значень. 3. Стиснення даних підтримується за допомогою Snappy 4. Підтримує ітерацію вперед / назад і пакетне написання 5. Використовується як бібліотека

Продовження таблиці 2.1

44.	Lightening Memory-Mapped Database (LMDB)	-	-	+	-	1. Безкоштовно 2. Вбудована база даних 3. Висока продуктивність 4. Надає прив'язку API для багатьох мов програмування. 5. Використовує багатоверсійний паралельний контроль пропонує високий рівень надійності
45.	Lotus Domino	+	-	-	-	1. Запатентована 2. Це багатозначна база даних
46.	Marklogic	+	+	-	-	1. Безкоштовно 2. Підтримує XML, JSON та RDF потрійні 3. Розподілений 4. Забезпечує високу доступність, повнотекстовий пошук, відповідність кислоті та безпеку
47.	Memcached	-	-	+	-	1. Безкоштовно 2. Система кешування пам'яті, яка є загальною і розподіленою 3. Масштабована архітектура 4. Підтримує шардінг
48.	MemcacheDB	-	-	+	-	1. Безкоштовно 2. Версія memcached, яка має стійкість 3. Це система кешування пам'яті, яка розподілена і має загальне призначення. 4. Розвиток зупинився на цьому рішенні.
49.	Microsoft SQL Server	-	+	-	-	1. Запатентована 2. Зазвичай використовується для моделювання взаємозв'язків багато-до-багатьох між даними 3. Інтеграція відносин здійснюється в Transact-SQL і основою СУБД є SQL Server
50.	MongoDB	+	-	-	-	1. Безкоштовно 2. Підтримує BSON або двійковий JSON 3. Дозволяє тиражування та шардування
51.	MUMP Database	+	-	-	-	1. Запатентована 2. MUMPS - це мова програмування із вбудованою базою даних 3. Використовується для додатків, пов'язаних із сектором охорони здоров'я
52.	Neo4j	-	+	-	-	1. Безкоштовно 2. Може використовуватися для транзакцій ACID 3. Підтримує кластеризацію та високу доступність 4. Забезпечує повну адміністративну підтримку

						5. Забезпечує вбудований REST API для інтерфейсу з іншими мовами програмування
53.	NoSQLz	-	-	+	-	1. Запатентована 2. Відповідає властивостям кислоти 3. Дозволяє операції CRUD (створення, читання, оновлення, видалення) 4. Простота в реалізації
54.	ObjectDatabase++	+	-	-	-	1. Запатентована 2. Бінарна 2. Структура рідного класу C ++
55.	OpenLink Virtuoso	+	+	+	-	1. Запатентована 2. Гібрид механізму баз даних та проміжного програмного забезпечення 3. Висока продуктивність і безпека 4. Підтримує SQL та SPARQL для виконання операцій над таблицями SQL та RDF 5. Підтримуються типи документів JSON, XML та CSV

Продовження таблиці 2.1

56.	Oracle NoSQL Database	-	-	+	-	1. Запатентована 2. Підтримує горизонтальну масштабованість та прозоре вирівнювання навантаження 3. Підтримує реплікацію та шардінг 4. Високодоступна і стійка до несправностей
57.	Oracle Spatial and Graph	-	+	-	-	1. Запатентована 2. Здатна обробляти RDF та графіки властивостей
58.	OrientDB	+	+	-	-	1. Безкоштовно 2. Підтримує JSON через HTTP 3. Підтримує використання мови типу SQL 4. Може використовуватися для транзакцій ACID 5. Підтримує шардінг, мульти-майстер-реплікацію, функції безпеки та режими без схем
59.	PostgreSQL	+	-	-	-	1. Безкоштовно 2. Підтримує JSONB, функцію JSON та JSON store 3. Підтримує HStore 2 і HStore
60.	Project Voldemort	-	-	+	-	1. Безкоштовно 2. Підтримує горизонтальну масштабованість 3. Доступність для операцій читання / запису висока 4. Відновлення несправностей є прозорим 5. Підтримує автоматичне розділення та реплікацію 6. Вважається придатним для додатків з інтенсивними операціями читання
61.	Qizx	+	-	-	-	1. Запатентована 2. Розподілена база даних XML 3. Підтримує текст, JSON та двійкові файли 4. Забезпечує інтегрований повнотекстовий пошук
62.	RavenDB	+	-	-	-	1. Безкоштовно 2. Повністю транзакційні та високі показники 3. Високодоступний 4. Багатоплатформна і проста у

						використанні 5. Багатомодельна архітектура, що дозволяє їй добре працювати з системами SQL
63.	Redis	-	-	+	-	1. Безкоштовно 2. Доступ для читання / запису є ефективним 3. Відмовостійка 4. Підтримує автоматичне розділення 5. Підходить для додатків, що включають структуровані рядки
64.	RethinkDB	+	-	-	-	1. Безкоштовно 2. Розподілена база даних 3. Підтримує JSON 4. Забезпечує заточування та тиражування
65.	Riak	-	-	+	-	1. Безкоштовно 2. Високодоступний та стійкий до несправностей 3. Високо масштабований та простий в експлуатації 4. Також доступні хмарні сховища та корпоративні версії Riak 5. Підтримує автоматичний розподіл даних та тиражування для стійкості та вдосконалення продуктивносты.
66.	RocketU2	+	-	-	-	1. Запатентована 2. Забезпечує динамічну підтримку 3. Масштабована 4. Надійний та ефективний 5. Підходить для управління діловою інформацією

Продовження таблиці 2.1

67.	RocksDB	-	-	+	-	1. Безкоштовно 2. Вбудована база даних, що забезпечує високу продуктивність 3. Підтримує всі функції LevelDB. Крім того, він також підтримує геопросторову індексацію, універсальне ущільнення, сімейства стовпців та операції.
68.	SAP HANA	+	+	-	-	1. Запатентована 2. Підтримує лише JSON 3. Може використовуватися для транзакцій ACID
69.	Scalaris	-	-	+	-	1. Безкоштовно 2. Високодоступна та стійка до несправностей 3. Масштабована 4. Послідовна 5. Самокерування 6. Мінімальні накладні витрати на обслуговування 7. Вважається придатним для програм, які інтенсивно читають / записують
70.	ScyllaDB	-	-	-	+	1. Безкоштовно 2. Розподілена 3. Призначена для інтеграції з Кассандрою для зменшення затримки та покращення пропускної здатності 4. Підтримує Thrift та CQL, протоколи також підтримується Кассандрою
71.	Sedna	+	-	-	-	1. Безкоштовно 2. База даних XML

72.	SimpleDB	+	-	-	-	1. Запатентована 2. Розподілена база даних 3. Використовується як веб-сервіс спільно з Amazon EC2 та S3 4. Забезпечує доступність і допуск розділів
73.	Solr	+	-	-	-	1. Безкоштовно 2. Пошукова машина, написана на Java 3. Підтримує індексацію в режимі реального часу, повнотекстовий пошук, інтеграцію баз даних, динамічну кластеризацію та багатофункціональну обробку документів 4. Забезпечує реплікацію індексу та розподілений пошук 5. Масштабована та відмовостійка
74.	Sparksee	-	+	-	-	1. Запатентована 2. Масштабована 3. Висока продуктивність 4. Перша база даних графіків для мобільних телефонів 5. Прив'язки, доступні для C ++, C #, Objective C, Python та Java
75.	Sqrrl	-	+	-	+	1. Запатентована 2. Розподілений 3. Масштабована 4. База даних у режимі реального часу 5. Забезпечує захист на рівні клітини
76.	Tarantool	-	-	+	-	1. Безкоштовно 2. Забезпечує стійкість до збоїв за допомогою 3. ведення журналів запису вперед 4. Може інтегруватися з іншими програмами та фреймворками, написаними різними мовами програмування.
77.	TokuMX	+	-	-	-	1. Безкоштовно 2. Версія MongoDB 3. Підтримує індексацію фрактального дерева

Продовження таблиці 2.1

78.	TerraStore	+	-	-	-	1. Безкоштовно 2. Зберігання в пам'яті 3. Динамічна конфігурація кластера 4. Стійкий 5. Підтримує балансування навантаження та автоматичний перерозподіл даних 6. Використовується для структурованих великих даних
79.	Tokyo Cabinet and Kyoto Cabinet	-	-	+	-	1. Безкоштовно 2. Надає дві бібліотеки для управління базами даних 3. Зберігання здійснюється за допомогою хеш-таблиць та дерев B + 4. Забезпечує обмежену підтримку транзакцій
80.	XAP	-	-	+	-	1. Запатентована 2. Програмна платформа для обчислень в пам'яті 3. Відповідні випадки використання цього рішення включають аналітику в реальному часі та обробку транзакцій, що вимагає низької затримки та високої продуктивності

						рівнів.

2.8. Формати файлів великих даних: Зберігання даних в екосистемі Hadoop

Hadoop пропонує один з найбільш економічно ефективних та ефективних способів зберігання даних у величезних обсягах. Більше того, структуровані, напівструктуровані та неструктуровані типи даних можна зберігати, а потім обробляти за допомогою таких інструментів, як Pig, Hive та Spark, щоб отримати результати, необхідні для будь-якого майбутнього аналізу чи візуалізації. Hadoop дозволяє користувачеві зберігати дані у своєму сховищі кількома способами. Деякі загальнодоступні та зрозумілі робочі формати включають XML, CSV та JSON.

Хоча JSON, XML і CSV - це зручні для читання формати, але це не найкращий спосіб зберігати дані на Hadoop. Насправді, в деяких випадках зберігання даних у таких необроблених форматах може виявитись дуже неефективним.

Більше того, паралельне зберігання даних для таких форматів неможливе. З огляду на той факт, що ефективність зберігання і паралелізм - дві головні переваги використання Hadoop, використання необроблених форматів файлів може просто перемогти ціле призначення.

CERN [29] обрав чотири технології-кандидати, ORC Parquet, Kudu та Avro для цієї мети. Однак ми включили до цього інші формати файлів даних, такі як Arrow та текстові формати дискусія для зрозумілості. 3 показує класифікацію та ієрархію форматів файлів великих даних. Кожен з форматів файлів має низку переваг та недоліків у назві. Цей розділ досліджує ці аспекти файлу великих даних форматів та забезпечить обговорення критеріїв, які необхідно використовувати для вибору формату файлу.

Текстові формати

Найпростіший приклад такого формату файлу даних - це записи, що зберігаються по черзі, закінчувані символом нового рядка або повернення карети. Для стиснення даних потрібно використовувати кодек стиснення на рівні файлу, такий як BZIP2. Три у цій категорії доступні формати, а саме текст або CSV, JSON та XML.

Звичайний текст

Дані, що зберігаються у цьому форматі, в основному форматуються таким чином, що поля мають або фіксовану ширину, або роздільник окремих записів, як у випадку CSV, у яких записи відокремлюються комами.

XML

Можна використовувати зовнішні визначення схем у XML. Однак виконання серіалізації та десеріалізації, як правило, погана.

JSON

JavaScript Object Notation (JSON) ефективніше, ніж XML, але проблема в ефективності серіалізації та десеріалізації існує.

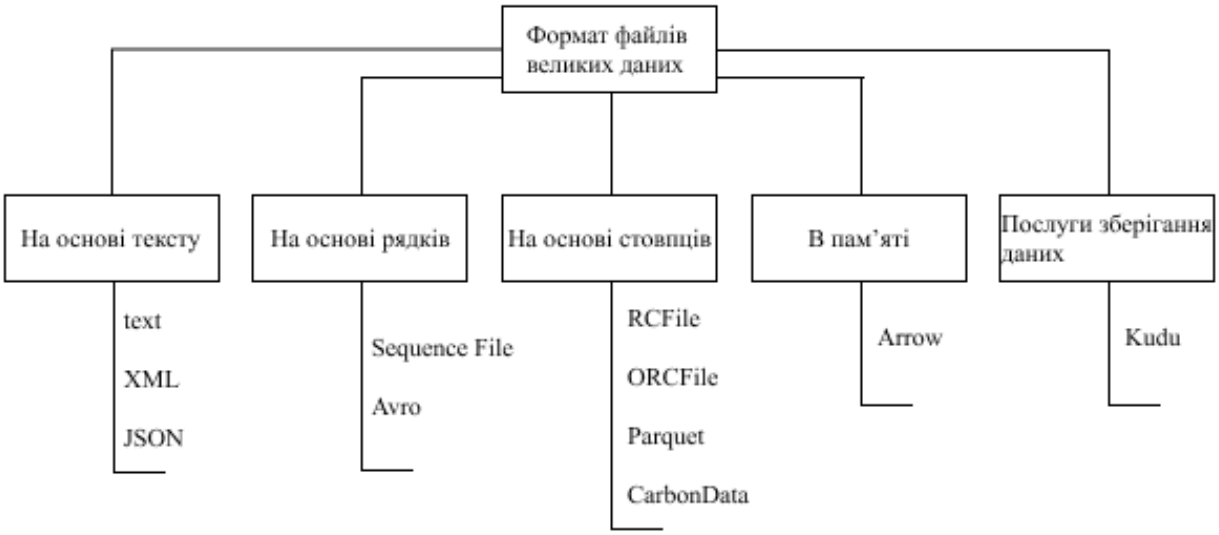

Рис.2.8. Класифікація форматів файлів великих даних

2.9. Формати на основі рядків

Файл послідовності

Цей формат файлу підтримується як частина фреймворку Hadoop, частиною якого великий файл має контейнер двійкового файлу пари ключ-значення для зберігання декількох файлів невеликого розміру. Отже, пари ключ-значення, що відповідають записам, є закодовані. Інтеграція формату файлу даних SequenceFile є плавною з Hadoop, враховуючи той факт, що перша була розроблена для другої.

Avro

Apace Avro використовується для компактного двійкового формату як стандарт серіалізації даних. Як правило, він використовується для зберігання постійних даних, пов'язаних з протоколами зв'язку та HDFS. Одна з головних переваг використання Apache Avro відрізняється високими показниками поглинання, що пояснюється швидкою та легкою десериалізацією та серіалізацією, передбачені тим самим. Важливо зазначити, що Avro не має внутрішнього індексу. Однак техніку розділення на основі каталогів, доступну в HDFS, можна використовувати для полегшення випадкового доступу до даних. Стиснення даних алгоритми, що підтримуються Apache Avro, включають DEFLATE та Snappy.

Існують інші рамки серіалізації та десеріалізації, такі як буфери економності та протоколу [30] конкуренція з Авро. Однак Avro є вбудованим компонентом Hadoop, хоча ці рамки є зовнішніми.

Крім цього, визначення схеми в Avro виконується за допомогою JSON, тоді як буфери Thrift і Protocol залежать від інтерфейсу мови визначення (IDL) [30] для визначення схеми.

2.10. Формати на основі стовпців

Формат файлу стовпчастого запису (RC)

Це найпримітивніший стовпчастий формат записів, який було створено в рамках проекту Apache Hive. RCFile [31] - це двійковий формат, подібний до SequenceFile, який забезпечує високе стиснення для операцій, що включають кілька рядків.

Стовпці зберігаються як запис стовпчастим способом, створюючи розділення рядків. Вертикальні перегородки створюються на розбиттях рядків стовпчастим способом. Метадані зберігаються для кожного розділеного рядка як ключ і зберігаються відповідні дані як його значення.

Оптимізований формат файлу стовпця (ORC)

Оптимізований стовпчик рядків [32] схожий на Parquet та RCFile в тому сенсі, що всі ці три формати файлів даних існують в рамках Hadoop. Продуктивність читання краща. Однак записи повільніші за середні.

Крім того, можливості кодування та стиснення цього формату файлу кращі, ніж у його аналогів, з ORC, що підтримує Zlib та Snappy.

Parquet

Apache Parquet [33] - стандарт серіалізації даних, який орієнтований на стовпці та, як відомо, покращує ефективність суттєво. Цей формат даних також включає в себе оптимізацію, як стиснення ряду значень, що належать до колони, що призводить до поліпшення коефіцієнтів ущільнення. Крім цього, такі кодування, як упаковка бітів, словник та запуск кодування довжини є додатковою оптимізацією. Для стиснення даних Snappy та GZip підтримуються алгоритми.

CarbonData

Цей формат даних був розроблений Huawei для управління наявними недоліками у вже доступних форматах.

CarbonData - відносно новий формат файлу даних, який дозволяє розробникам скористатися перевагами орієнтованих на стовпці зберігання, а також надає підтримку для обробки запитів довільного доступу. Дані згруповані в блоккети, що є зберігається поряд з іншою інформацією про такі дані, як схема, індекси та зсуви, крім інших. Метадані є зберігається у

верхньому та нижньому колонтитулах, що пропонує значну оптимізацію продуктивності під час сканування та обробки подальші запити.

Формати в пам'яті

Apache Arrow [34] - це платформа для розробки програм, що використовують дані в пам'яті. Більше того, це працює по всьому мови, що робить його стандартом для стовпчастого формату пам'яті, що забезпечує підтримку як ієрархічної, так і плоскої мови даних. Дані організовані для забезпечення високопродуктивної аналітики сучасного обладнання. Міжпроцесовий зв’язок потокового передавання працює на нульовій копії або не має десеріалізації та серіалізації. Крім цього, він пропонує багато обчислювальні бібліотеки для складних задач.

2.11. Послуги зберігання даних та порівняння форматів файлів великих даних

Kudu [35] - це система зберігання, яка забезпечує масштабованість і базується на концепції розподіленого сховища. Дані зберігаються всередині таблиць. Більше того, цей формат даних забезпечує оптимізований компроміс між продуктивністю та швидкістю проковтування, яке приписується стовпчастій організації даних та підтримці індексу. Стиснення даних алгоритми, що підтримуються Apache Kudu, включають LZ4, Zlib та Snappy.

Порівняння форматів файлів великих даних

Фактичне зберігання даних у файловій системі визначається обраним форматом файлу даних, що є вирішальним вибором зробити з огляду на той факт, що він відіграє значну роль в оптимізації системного зберігання. Рішення про вибір файлу формату програми залежить від випадку використання або алгоритму, що використовується для обробки даних. З урахуванням цього вибраний формат файлу повинен відповідати деяким основним вимогам, щоб вважатись придатним для систем великих даних.

По-перше, вибраний формат файлу великих даних повинен бути виразним і чітко визначеним. По-друге, вона повинна бути різноманітною можливості обробки, що стосується підтримуваних структур даних. Деякі з основних структур, які повинні бути підтримувані включають структури, карти, числа, рядки, записи та масиви. Нарешті, формат файлу великих даних має бути двійковим, простим і забезпечувати підтримку стиснення.

Одне з найбільших вузьких місць у програмах, пов'язаних з HDFS, які використовують такі технології, як Spark та Hadoop включає зменшення часу читання та запису даних. Проблеми, такі як обмеження на зберігання, еволюціонуючі схеми та великі дані ще більше ускладнюють системні вимоги. Для того, щоб пом'якшити ці проблеми між доменами додатків та проблемних сценаріїв з’явилося кілька форматів файлів великих даних.

Використання відповідного формату файлу може скористатися системою наступними способами:

1. Час читання скорочується.

2. Час запису скорочується.

3. Файли можна розділити, що іншими словами означає, що більше не потрібно читати весь файл отриманням його меншого підрозділу.

4. Існує підтримка розвитку схеми, і схема може бути змінена за запитом залежно від зміни потреби системи.

5. Доступні вдосконалені кодеки стиснення, що гарантують стиснення файлів без втрати переваги базового формату.

З огляду на вищезазначені переваги, вибір правильного формату файлу даних може оптимізувати продуктивність системи істотно. Однак існує безліч варіантів щодо цього. Хоча деякі формати файлів розроблені для загального користуванні є деякі інші, які пропонують переваги оптимізації для конкретних програм або вдосконалюють конкретні характеристики. Це робить порівняння форматів файлів, необхідних для полегшення рішення, який файл даних формат найкраще підходить для програми. У таблиці 2.2 узагальнено переваги, недоліки та типові випадки використання для різних форматів файлів даних, обговорені в попередніх розділах.

Таблиця 2.2

Порівняння форматів файлів данних

Клас	**Формат файлу даних**	**Переваги**	**Недоліки**	**Відповідність використання**
Формати текстових файлів даних	Text XML JSON	1. Невелика вага	1. Читання повільне. 2. Писати повільно. 3. Місце витрачається даремно через непотрібні заголовки стовпців. 4. Стиснуті файли не можна розділити, що веде до величезних карт. Більше того, на рівні блоку стиснення не підтримується.	Підходить для використання структурованих даних на HDFS. Також файли CSV використовуються у випадках, коли дані потрібно витягувати з Hadoop та масово завантажувати у базу даних.

Продовження таблиці 2.2

Формат файлу даних на основі рядків	Файл послідовності	1. Цей формат файлу компактний у порівнянні з текстовими файлами. 2. Формат файлу підтримує додаткове стиснення. 3. Він підтримує паралельну обробку. 4. Величезна кількість файлів невеликого розміру може зберігатися у контейнері, наданому з тією ж метою. 5. Однією з найбільших переваг цього формату файлу даних є підтримка стиснення на рівні блоків, що дозволяє стиснення файлів, одночасно дозволяючи розділення файлів для кількох завдань.	1. Цей формат файлу не є кращим для таких інструментів, як Hive. 2. Функціональність додавання файлового формату так само хороша і порівнянна з іншими форматами файлів. 3. У форматі файлу відсутня підтримка декількох мов.	Якщо проміжні дані, які генеруються між робочими місцями, потрібно зберегти, використовується формат файлу даних SequenceFile (файл послідовності).
	Avro	1. Розмір серіалізованих даних є найменшим. 2. Він пропонує стиснення на рівні блоку, що дозволяє одночасно розділяти файли. 3. Цей формат файлу підтримує структуру об’єкта. 4. Навіть якщо схема змінилася, Avro дозволяє читати старі дані. 5. Визначення схеми записані у форматі JSON. Тому розробка значно спрощується в мовах програмування, які мають JSON бібліотеки.	1. Процеси читання та запису потребують визначення схеми.	Avro вважається найкращим у тих випадках, коли еволюція схеми є ключовою вимогою. Якщо схема, як очікується, буде змінюватися з часом, тоді Avro є кращим. Насправді Avro є кращим у всіх випадках використання Hadoop.
На основі стовпців Формат файлу даних	RCFile	RCFile пропонує типові переваги пов'язані зі стовпчастими базами даних, які включають: 1. Хороше стиснення. 2. Ефективність запитів краща, ніж для баз даних, орієнтованих на рядки.	1. Процеси читання та запису вимагають визначення схем. 2. Немає підтримки схеми еволюція. 3. Процес запису відбувається повільніше.	Якщо випадок використання передбачає таблиці, що мають багато стовпців і додаток вимагає частого використання певних стовпців, тоді RCFile є найкращим форматом даних.
	ORCFile	1. Можливості стиснення ORCFile кращі, ніж у RCFile. 2. Обробка запитів також покращується в порівнянні з RCFile.	1. Немає підтримки еволюції схем. 2. Формат ORCFile погано підтримується Apache Impala.	ORCFile та Parquet використовуються у сценаріях, коли продуктивність запиту має вирішальне значення. Однак було виявлено, що паркет при використанні із SparkQL демонструє найкращі покращення у виконанні запитів.
	Parquet	1. Як і у випадку з форматом ORCFile, стиснення та ефективність запитів є хорошими. Більше того, у випадках, коли запитуються конкретні стовпці, цей формат файлу даних є особливо ефективним. 2. Ефективність читання хороша. 3. Еволюція схеми, у цьому випадку, краща за формат ORCFile, оскільки стовпці можуть бути	1. Писання інтенсивно обчислювальні. 2. Якщо програма вимагає рядків даних, то, як і всі стовпчасті бази даних, паркет може не мати ефективної продуктивності з огляду на затрати на мережеві дії.	

		додані в кінці. 4. Оптимізація сховища є чудовою, і вона пропонує як рівень файлів, так і рівень блоку стиснення.		

Продовження таблиці 2.2

	CarbonData	1. Він підтримує операції оновлення та видалення, що має вирішальне значення для багатьох робочих процесів. 2. CarbonData пропонує такі оптимізації, як сегментування та багаторівнева індексація. Отже, об'єднання двох файлів є більш ефективним, а запити швидші, ніж ніколи.	1. CarbonData не підтримує кислоту. 2. Розмір стиснених файлів більше, ніж у ORC та Parquet. 3. CarbonData - відносно новий формат файлу даних, оскільки багато технологій, таких як Athena та Presto, ще не підтримують його.	CarbonData може використовуватися для змінних аналітичних навантажень із типовими випадками використання, що включають інтерактивні або докладні запити та запити, що включають реальний час поглинання та агрегування / OLAP BI.
Формат файлу даних в пам'яті	Arrow	1. Цей формат дозволяє системам обробляти великі масиви даних. 2. Одну і ту ж пам'ять можуть спільно використовувати різні програми за допомогою загального рівня доступу до даних. 3. Цей формат файлу оптимізований для паралельної обробки та локалізації даних. Більше того, він розроблений для множинних даних з однією інструкцією (SIMD). 4. Це дозволяє обробляти сканування, а також навантаження з довільним доступом. 5. Накладні витрати, пов'язані з потоковими повідомленнями та RPC, значно зменшуються. 6. Можна використовувати стрілку Apache з графічними процесорами. 7. Стовпчастий формат, наданий Apache Arrow, є _Подроблені 'та підставки вкладені та плоскі схеми.	1. На сьогодні реалізація предикатного натискання залишається за двигуном. Хоча, як очікується, Apache Arrow зможе використовувати багаторазові швидкі векторизовані операції, але зусилля в цьому напрямку ще не оформилися.	Стрілка Apache найкраще підходить для векторизованої обробки, яка передбачає єдину інструкцію щодо множинних даних (SIMD).
Послуги зберігання даних	Kudu	1. Навантаження OLAP можна швидко обробити за допомогою Kudu. 2. Kudu можна легко інтегрувати з Spark, Hadoop та його екосистемою. 3. Його можна щільно інтегрувати з Apache Impala, який є ефективною альтернативою паркету з HDFS.	1. У Kudu немає вбудованих параметрів відновлення та резервного копіювання даних. 2. Існують деякі обмеження безпеки, такі як авторизація, доступна лише на системному рівні, і відсутність вбудованої підтримки шифрування даних. 3. Kudu не підтримує	Kudu створений для додатків, орієнтованих на дані часових рядів, і для таких програм, як онлайн-звіти та аналіз даних машини.

		4. Система дуже доступна. 5. Надана модель даних структурована. 6. Управління та адміністрування Kudu є простим. 7. Модель узгодженості є гнучкою та сильною. 8. Випадкові та послідовні робочі навантаження можуть виконуватися одночасно з висока продуктивність.	автоматичне розділення та переділ даних. 4. Існують інші обмеження на рівні схеми, такі як відсутність підтримки вторинних індексів та багаторядкових транзакцій.	

З порівняльної таблиці можна зробити висновок, що хоча текстові формати є простими та легкими, вони представляють безліч недоліків, які можуть суттєво вплинути на продуктивність системи. Для того, щоб подолати обмеження текстових форматів, Hadoop має вбудовані формати файлів даних. Перший із цих форматів - це дані на основі рядків формат файлу під назвою SequenceFile (файл послідовності) [36]. Інші формати файлів даних, які базуються на SequenceFile та включені до екосистеми Hadoop включає MapFile, SetFile та BloomMapFile. Ці формати файлів призначені для спеціалізованих випадків використання. SequenceFile підтримує лише Java, що робить його залежним від мови та не підтримує версій. Як результат, Avro, який переважає SequenceFile, виявився найпопулярнішим на основі рядків форматом файлу даних. Зрозуміло, що Avro є найкращим варіантом серед форматів файлів даних на основі рядків.

У випадку форматів файлів, орієнтованих на рядки, у файлі здійснюється суцільне зберігання рядків. З іншого боку, у випадку орієнтованих на стовпці форматів, рядки розділяються, а значення для розділення рядків зберігаються по стовпцях. Наприклад, значення для першого стовпця рядка щілина зберігається спочатку тощо. Отже, стовпці, не потрібні для обробки запиту може бути опущено під час доступу до даних. Іншими словами, формати, орієнтовані на рядки, найкраще підходять для випадків, коли менше рядків, які потрібно прочитати, але для цього рядка потрібно багато стовпців. З іншого боку, якщо є невелика кількість стовпців тоді потрібні формати файлів, орієнтовані на стовпці.

Важливо зазначити, що для форматів файлів даних, орієнтованих на стовпці, потрібен буфер для розділення рядків, оскільки він працює більше одного рядка одночасно. Більше того, неможливо контролювати процес написання. Якщо процес не вдається, не можна відновити поточний файл. Це причина, чому формати, орієнтовані на стовпці, не використовуються для потокового передавання. З іншого боку, використання Avro дозволяє читати до того моменту, до якого відбулася синхронізація. Флюм робить використання форматів, орієнтованих на рядки, завдяки цій властивості [37].

Системи розроблені таким чином, що прагнення до диска зводиться до мінімуму і, отже, затримка є зменшеною. Це ефективний механізм зберігання транзакційних робочих навантажень, для яких дані записуються по рядках. Для аналітичного навантаження, велика кількість рядків повинна бути доступна одночасно. Однак підмножина стовпців можливо, доведеться прочитати для обробки запиту. У таких сценаріях формат, орієнтований на рядки, неефективний, враховуючи той факт що всі стовпці з декількох рядків доведеться прочитати, навіть якщо всі ці стовпці не потрібні. В використанні очікується, що орієнтований на стовпці формат зменшує кількість пошуків, покращуючи продуктивність запитів. Хоча, в цьому випадку повільніші, але для аналітичних навантажень це, як очікується, буде працювати добре, оскільки кількість зчитувань, їх кількість перевищить кількість записів.

RCFile [27] - це найбільш примітивний формат файлу даних на основі стовпців, а ORCFile - це оптимізована версія. Хоча ORCFile [32] вважається придатним для додатків з транзакціями ACID і пропонує швидкий доступ з його функцією індексування, Parquet просувається Cloudera і, як відомо, найкраще працює з Spark. Найдосконалішим та найновішим форматом файлів у цій категорії є CarbonData [38] але завдяки новому статусу сумісність з технологіями сумнівна. Це робить Parquet найпопулярнішим форматом файлу даних на основі стовпців.

Arrow і Kudu [35] підтримують стовпчасте представлення з тією різницею, що Arrow підтримує “in-memory” зберігання, в той час як Kudu

забезпечує сховище, яке можна змінювати на диску проти формату Parquet, який незмінний на диску і на дисковому сховищі. Компроміси за використання незмінного формату файлу даних (Parquet) включають більшу пропускну здатність для запису, простіше одночасний спільний доступ, доступ і тиражування, а також відсутність накладних витрат, пов'язаних з операціями. Однак для виготовлення будь-яких модифікацій, набір даних потрібно переписати. Змінюваний (Kudu) забезпечує більшу гнучкість у компромісі між швидкістю оновлення та читання. Час затримки короткого доступу нижчий через реплікацію кворуму та індексація первинного ключа. Більше того, семантика подібна до семантики баз даних. З урахуванням цього окремий демон це необхідність для управління.

Порівнюючи пам'ять Parquet на диску з тимчасовою пам'яттю Arrow, важливо зауважити, що перший дозволяє отримати доступ до декількох запитів з пріоритетом, що надається зменшенню вводу-виводу, але пропускна здатність процесора повинна бути доброю, однак сховище на диску вважається придатним лише для потокового доступу. З іншого боку, в пам'яті сховище підтримує потокове передавання, а також довільний доступ і орієнтоване на виконання одного запиту. У цьому випадку пропускна здатність процесора є пріоритетною, хоча пропускна здатність вводу-виводу також повинна бути хорошою. Деякі проекти використовують Arrow і Parquet разом. Pandas [39] - один із прикладів такого використання. Кадр даних від Pandas можна зберегти на паркет, який потім можна прочитати на стрілці. Здатність Pandas працювати на колонах Arrow дозволяє це легко інтегрувати з Spark.

2.12. Майбутні тенденції технології зберігання великих даних

Технології, що використовуються для зберігання великих даних, з часом дозріли, щоб поступитися місцем таким технологіям, як блокчейн,

який продемонстрував значне зростання популярності та використання в широкій колекції доменів [40]. Як є у випадку з будь-якою технологією, яка використовується в різних доменах, у ній було визначено кілька проблем та проблем додатки до реальних сценаріїв. Однією з найглибших проблем цієї технології є масштабованість.

Однак здатність цієї технології вивести Інтернет на наступний етап шляхом створення децентралізованої мережі робить це вартим обговорення.

Одним з фундаментальних компонентів децентралізованої мережі є децентралізоване сховище [41]. На додаток до блокчейн, деякі інші технології, такі як Інтернет речей (IoT) та штучний інтелект, також є випробувальні полігони існуючих рішень для зберігання даних та їх можливості. Піднесення Інтернету речей (IoT) та його очікується, що покриття 20 мільярдів підключених пристроїв до 2020 року потребує придбання, зберігання, управління та обробка величезних резервів даних.

Попит на рішення для зберігання даних зростає з огляду на проблеми, які постає перед управлінням даними підключені пристрої, обмін даними та необхідність персоналізації в додатках [42]. Відбувається зміна підходу до інтенсивно використовуваних даних, особливо з огляду на залежність компаній від купи даних. Однак, ці дані зберігаються в централізованих центрах обробки даних, що призводить до численних порушень безпеки та проблем [43]. Цей аргумент підтримує використання децентралізованих рішень для зберігання даних.

Однак побудова децентралізованих додатків поверх технологій блокчейну пропонує свій набір незалежних викликів. Наприклад, управління даними та їх зберігання не підтримуються такими рішеннями, як Ethereum.

Тому, коли ці рішення змушені виконувати свої можливості, вони споживають більше простору та більше часу. Бачення децентралізованого зберігання - об’єднати особливості технології блокчейну з рішеннями, що можуть задовольнити практичні вимоги зберігання великих даних. З самої назви видно, що розподілене сховище ділить дані на менші одиниці даних і

розподіляє їх по різних вузлах, де вони зберігаються. Ця характеристика може бути порівняно з технологією розподіленої книги [44], яка зазвичай асоціюється з блокчейном. В даний час навіть хмарні бази даних занадто централізовані і їх легко орієнтувати хакери [45]. Більше того, пункти поразки дуже очевидні, що робить їх тим більше, що їх легше атакувати. Відключення електроенергії є однією з таких точок відмови [46]. Навпаки, децентралізована система позбавлена таких проблем з огляду на той факт, що вузли є географічно віддалені та фізично не пов’язані. Отже, очікується відключення або атака на місці відмови, що впливає лише на відповідний вузол. Загалом система залишатиметься незмінною, як і інші вузли системи продовжуйте функціонувати як зазвичай. Окрім того, що робить систему надійною та доступною, децентралізація також сприяє до масштабованості та продуктивності системи [47].

Проводячи паралелі між децентралізованим сховищем та блокчейном, важливо усвідомити, що зберігання блокчейн залишається важливою проблемою. Оскільки все більше транзакцій відбувається на блокчейні, масштабованість може ускладнитися. Тому зберігання великих даних, безумовно, є сумнівним доменом блокчейн перспектива. Для пом'якшення вищезазначених проблем використовуються такі методи, як роїння та використовується шардінг. Розбиття баз даних за логічними лініями називається шардінгом. Децентралізована модель забезпечує зберігання осколків разом. Більше того, унікальний розділовий ключ використовується спеціальним децентралізований додаток для доступу до осколків. Крім цього, роїння використовується для колективного зберігання осколків.

Дані зберігаються та управляються шляхом створення великої групи вузлів, яка називається роєм. Ця група вузлів є подібною до мережі вузлів, створених для блокчейну.

Найглибшими перевагами роїння є зменшення затримки та збільшення швидкості, враховуючи той факт що дані зазвичай отримуються з найшвидшого та найближчого вузлів. На додаток до цього, вузли

розташовані географічно віддалений, що робить систему масштабованою та надійною. Ще одним важливим аспектом децентралізованого зберігання з точки зору замовника полягає в тому, що замовник матиме можливість придбати сховище на рівні вузла у різних постачальників проти нав'язливого одного продавця. Це можна розглядати як важливий крок на шляху до полегшення прийняття цього технологія в реальних сценаріях. Можна зробити висновок, що децентралізоване сховище, безсумнівно, запропонує масштабоване, ефективне та безпечне рішення для майбутнього зберігання великих даних.

Висновки

1. Було наведено класифікацію рішень NoSQL на основі підтримуваної моделі даних, яка може бути орієнтована на дані, графік, ключ-значення або широкий стовпець.

2. Проаналізовано 80 рішень NoSQL з урахуванням критеріїв вибору технологій для систем Великих даних поряд із сукупною оцінкою відповідних та невідповідних випадків використання для кожної моделі даних.

3. Наведено класифікацію форматів файлів великих даних за п’ятьма категоріями, а саме на основі тексту, рядків, стовпців, в пам'яті та послуги зберігання даних.

4. Порівняно доступні формати файлів даних, аналізуючи переваги та недоліки та випадки використання кожного файлу даних формат.

РОЗДІЛ 3

ПРАКТИЧНЕ ЗАСТОСУВАННЯ НА ПРИКЛАДІ РОБОТИ З МЕДИЧНИМИ ДАНИМИ

3.1 Опис інформаційної моделі

Результат кожного обстеження – це заповнений медичним працівником протокол дослідження. Всі дані можна розділити на три категорії:

• множина вимірювань;

• інформація про візит до лікаря;

• множина мультимедійних матеріалів (графіки, схеми, різні зображення).

Множина вимірювань та інформація про візит до лікаря призначені для опису досліджуваного об'єкта - анатомії. Однак наша сутність анатомії має ширше значення, ніж анатомія в біологічному сенсі. Вона включає в себе частини тіла (голова, шия і т.п.), органи (печінка, серце і т.п.), тканини, продукти життєдіяльності організму (сеча, кал), кістки, рідини, що є діяльністю будь-яких органів (грудне молоко, секрет передміхурової залози), різні новоутворення, слизові оболонки, мікроорганізми та інше. З одного боку, всі об'єкти входять в різні системи - функціональний підхід. З іншого боку, можна розглянути дані в структурному зрізі. Саме так виникає ієрархія анатомії – досліджуваний об’єкт.

Інформація про візит до лікаря – це набір текстових фраз, що включають у себе скарги пацієнта, стан досліджуваного об'єкта, діагноз, рекомендації до лікування. Найголовніше, що інформація про візит дозволяє висловити свою думку лікаря для надання більш повної картини іншим колегам і пацієнту.

Множина вимірів, наприклад, лабораторні дослідження, кожне з яких можуть мати текстове значення та числове (діапазони значень), оскільки багато параметрів (наприклад, розміри органів, частота серцевих скорочень, кількість наявних речовин) залежить від таких даних як вік, стать, ріст, вага, фаза циклу, вагітність, куріння, час взяття аналізу. Для числового значення використовується референтний інтервал. Референтний інтервал – це межі норми.

Множина мультимедійних матеріалів займає значний обсяг в медичних інформаційних системах та є важливим джерелом даних при діагностиці, оцінці і плануванні терапії. Наприклад, комп'ютерна томографія, магнітно-резонансна томографія, рентген, молекулярна візуалізація, ультразвукове дослідження, Фотоакустична томографія, рентгеноскопія, позитронно-емісійна томографія з комп'ютерною томографією.

Рис. 3.1. Інформаційна модель

3.2 Документо-орієнтована база даних

Сімейство нереляційних моделей даних поділяють на чотири види: ключ-значення, документо-орієнтовані, стовпчикові і графові бази даних. Проаналізувавши кожний вид, для медичних карт було обрано документо-орієнтовану базу даних.

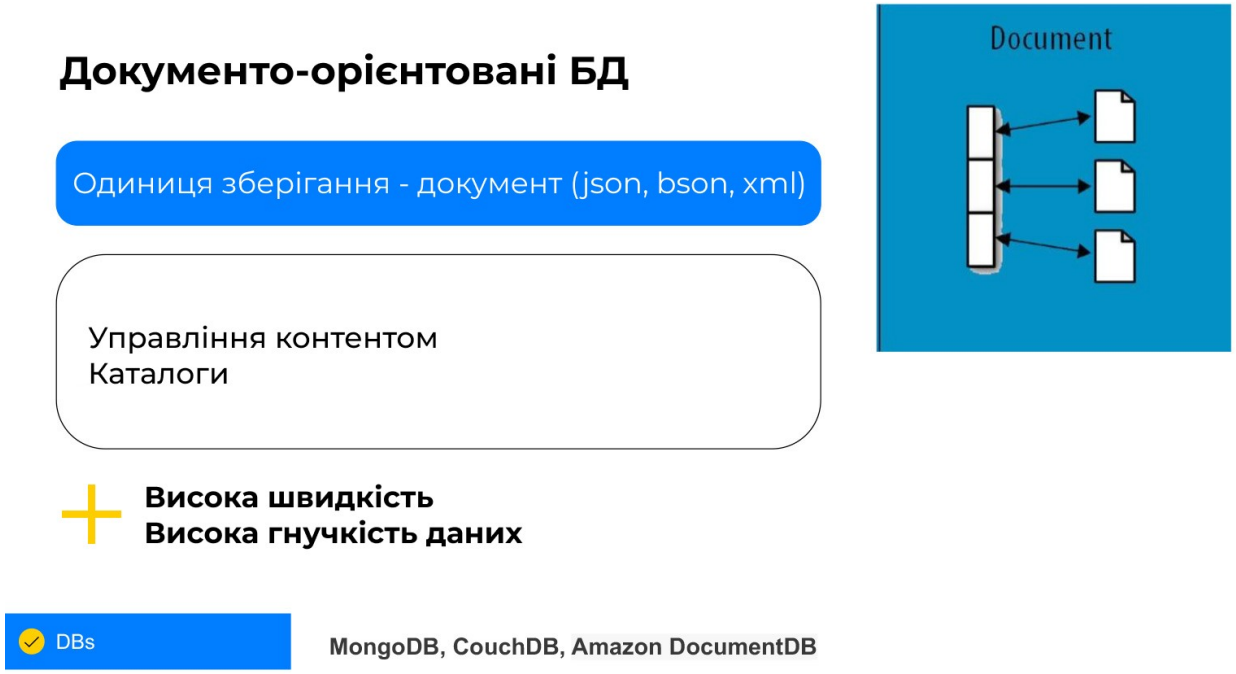

Рис. 3.2 Документо-орієнтована база даних

База даних NoSQL, яка використовує документи для зберігання та пошуку даних, називається документо-орієнтованою базою даних[48]. Інші назви цієї моделі даних NoSQL - це база даних документів [49] та сховище документів [50]. Вона насамперед використовується для управління напівструктурованими даними через гнучкість та підтримку змінної схеми. Документо-орієнтовані бази даних дуже схожі на ключ-значення у деяких із областей їх застосування. Але у них одиницею є документ.

Кожен документ може зберігати в собі, як правило, XML, JSON або BSON - бінарному-зберігається JSON. Але зараз практично завжди JSON або BSON.

У будь-який документ, в будь-який JSON ви можете записати абсолютно будь-який набір даних. І вони дуже часто застосовуються - наприклад, коли потрібно зробити який-небудь каталог і коли кожен продукт в каталозі може мати різні характеристики.

Наприклад, профілі користувачів. Хтось вказав свій улюблений фільм, хтось - улюблену їжу. Щоб не засовувати все в одне поле, яке буде зберігати незрозуміло що, ми можемо все записати в JSON документо-орієнтованих бази даних.

Документи групуються разом, утворюючи структуру, яка називається колекцією [5]. Може бути кілька колекцій баз даних. Ця структура подібна до функціонування таблиці, яка присутня в реляційній базі даних.

Бази даних, орієнтовані на документи, забезпечують механізм виконання запитів до колекцій та отримання документів, які задовольняють вимоги до атрибутів. Існує кілька переваг використання цього підходу, які включають:

1. Більша частина зростаючих даних надходить з пристроїв IoT, соціальних медіа та Інтернету. Орієнтовані на документи бази даних пропонують гнучке моделювання даних [5], на відміну від реляційних баз даних, які змушують програми вбудовувати дані в існуючі моделі незалежно від їх потреби.

2. Ефективність записування баз даних, орієнтованих на документи, краща за звичайні системи. Щоб зробити систему доступною для написання, база даних може також порушити консистенцію даних. Отже, навіть якщо система виходить з ладу і реплікація займає більше часу, ніж очікувалося, операція запису буде швидкою.

3. Відомо, що функції індексації та механізми запитів баз даних, доступні в цій категорії, є швидкими та ефективними [5]. Тому вони пропонують більш швидку продуктивність запитів.

Вимоги до бази даних

Крім основних загальноприйнятих вимог до БД слід вивести власні додаткові вимоги, яким повинна задовольняти БД і відповідно СУБД керуюча їй.

● Медична картка може мати різну кількість полів, оскільки кожна людина матиме різний набір метаданих.

● Має бути можливість додавання нових "для користувача" полів в медичну картку. Тобто має бути присутня можливість простого розширення набору метаданих випробуваного.

3.3. MongoDB

У якості СУБД вирішено вибрати MongoDB. MongoDB - це крос-платформова документо-орієнтована система управління базами даних з відкритим кодом.

Вибір саме документо-орієнтованої СУБД заснований на тому, що на відміну від реляційних аналогів тут не накладаються обмеження на набір полів у документа. Так як документо-орієнтовані бази даних ще не так поширені як реляційні, але їхня популярність із кожним роком все збільшується, доцільно буде роз'яснити деякі нюанси термінології документо-орієнтованих баз даних.

У короткому підручнику з MongoDBTheLittleMongoDBBook від автора KarlSeguin [51] наведені шість основних концепцій MongoDB:

1. MongoDB - концептуально те ж саме, що звичайна, звична нам база даних (або в термінології Oracle - схема). Усередині MongoDB може бути нуль або більше баз даних, кожна з яких є контейнером для інших сутностей.

2. База даних може мати нуль або більше «колекцій». Колекція є аналогом таблиці з реляційних БД. Колекції складаються з нуля або більше «Документів». Документ є аналогом записи в таблиці (рядки).

Документ складається з одного або більше «полів», поля, в свою чергу, є аналогом "колонки".

3. «Індекси» в MongoDB майже ідентичні таким в реляційних базах даних.

4. «Курсор» відрізняються від попередніх п'яти концепцій, але вони дуже важливі

Важливо розуміти, що коли ми запитуємо у MongoDB будь-які дані, то вона повертає курсор, з яким ми можемо робити все що завгодно - підраховувати, пропускати певну кількість попередніх записів - при цьому не завантажуючи самі дані.

Основна ж відмінність документо-орієнтованих баз даних від реляційних є те, що в реляційних базах даних «колонки» визначаються на рівні «таблиці», тоді як документо-орієнтованих бази даних визначають «поля» на рівні «Документа». Тобто будь-який документ всередині колекції може мати свій власний унікальний набір полів, що і є ключовим аспектом, що робить документо-орієнтовані бази даних найбільш підходящими для даного завдання.

Вибір припав саме на MongoDB через її популярність та наявність хорошої і повної документації. Основними плюсами MongoDB для даного проекту, крім її документо-орієнтованості, стали:

● схожий з реляційними СУБД підхід до зберігання даних: базаданнихколлекція-документ-поле. Наприклад, у основного конкурента MongoDB, CouchDB, відсутнє поняття будь-якого поділу документів на групи і всі документи є рівнозначними, а для поділу документів, наприклад, по типу, необхідно додавати спеціальні поля.

● формат зберігання даних. Дані в MongoDB зберігаються в форматі BSON.

BSON - це практично те ж саме, що і JSON (поширений формат представлення даних в web), але в бінарному вигляді. Той факт, що дані зберігаються саме в бінарному вигляді, значно прискорює час їх обробки.

● існує інструмент GridFS, що дозволяє зберігати в БД файли будь-якого розміру. Варто зазначити, що на даному етапі розробки, файли великих розмірів, такі, наприклад, як DICOM-файли, пропонується зберігати в іншій, більш захищеній і призначеній для цього БД, під керуванням спеціалізованої СУБД, наприклад Oracle. Наявність такого інструменту є величезним плюсом.

3.4.1. Представлення даних в БД

Так як в якості СУБД була обрана документо-орієнтована MongoDB, то і структура бази даних буде приводиться в її термінології. Опис, для більшої наочності і розуміння, приведено в порядок розгортання від основних документів до побічних.

Також приведено часткове вирішення проблеми локалізації основних елементів данного формату. В кінці наведені лістинги з моделями основних документів з коментарями.

Основними документами в базі даних є: "Пацієнт" ("Testsubject"), "Лікарь" ("Researcher") і "Дослідження" ("Research"). Всі документи зберігаються в своїй окремій колекції, відповідно "Пацієнт", "Лікарь", "Дослідження". На рисунку 3.3 наочно показані зв'язки між трьома основними об'єктами бази даних. Всі документи зберігаються в своїй окремій колекції, відповідно "Пацієнти", "Лікарі", "Дослідження".

Рис. 3.3 Зв’язки між основними об’єктами

Лікарь

Документ "Лікарь" є аккаунтом реального наукового співробітника і містить в собі:

● унікальний ідентифікатор

● загальні дані самого наукового співробітника: ПІБ, вік, стать, лабораторія в якої він працює і т.д.

● обладнання і методи обстежень, які співробітник може використовувати

● поточний статус зайнятості: зайнятий, вільний і т.д.

● роль лікаря, яка регламентує права доступу, якими він володіє

● прапори бази даних: видалений / не видалений, дата створення документа (акаунту), дата останньої зміни

Пацієнт

Документ "Пацієнт" не має ніяких прав доступу і є відображенням медичної картки пацієнта, тобто лише зберігає набір метаданих кожного реального пацієнта для використання їх в ході різних досліджень. Не представляється можливим описати повну структуру даних даного об'єкта, так як він може містити в собі різний набір секцій, в яких зберігаються різні дані пов'язані з ним. Можна лише виділити набір секцій який буде обов'язковий для кожного екземпляра документа "Пацієнта":

● унікальний ідентифікатор

● загальні дані пацієнта: ПІБ, стать, вік. При анонімному опитування ПІБ відсутня

● прапори бази даних: видалений / не видалений, дата створення документа, дата останньої зміни і т.д.

Дослідження

Документ "Дослідження" є сполучною ланкою, що надають "Лікарям" доступ до даних певного набору "Пацієнтів". Тільки через нього більшість наукових співробітників ("Лікарів" в БД) зможуть отримати доступ до даних "Пацієнта", виняток можуть становити лише "Лікарі", які мають пріоритетні права доступу до колекції "Пацієнти". Документ "Дослідження" містить в собі:

● унікальний ідентифікатор

● загальну інформацію про дослідження: назва, мета, ініціатор (лабораторія або співробітник)

● список "Лікарів" працюють над дослідженням

● список "Пацієнтів" беруть участь в дослідженні

● прапори бази даних

Секція

Для зручності структурування даних зберігаються всередині основних об'єктів, використовуються секції. Структура документів в MongoDB дозволяє зберігати документи всередині документів, такий підхід в MongoDB називається денормалізованна модель даних.

"Секція" є окремим документом і зберігається всередині основних документів ("Пацієнт", "Лікарь", "Дослідження") або ж простіше кажучи, основні документи складаються з "секцій".

Важливо відзначити той факт, що основні документи не посилаються на свої "Секції", а кожна "Секція" зберігається саме всередині свого основного документа. Також варто помітити, що для зберігання списку "Лікарів" і "Пацієнтів" в "Дослідження" використовуються масиви (даний

тип даних також присутній в MongoDB), в яких вже як раз будуть зберігатися посилання на відповідні об'єкти.

"Пацієнт" має різний набір секції. Додаткові секції створюються користувачем (науковим співробітником) у якого є права для додавання нових і зміни даних вже існуючих "Пацієнтів".

Для того, щоб зберігати створені користувачем нові типи секцій і виключити створення дублікатів, створюється колекція секцій. Точніше кажучи колекція типів секцій, в якій зберігаються "скелети" вже створених секцій. У лістингу 2.1 приведена структура документа "Секція" в форматі JSON (в подібному вигляді вона зберігається в MongoDB), в коментарі наведені пояснення основних полів.

```
        {
_id : <Section-type-ID>, // ідентифікатор типу секції, задається явно при додаванні
sec_data : {             // дані секції також зберігаються як окремий документ всередині неї
data1 : <Some-Data>,
data2 : <Some-Data>, // дані можуть мати будь-який доступний в MongoDB тип, включаючи масиви
data3 : <Some-Data>,
        ...
dataN : <Some-Data>
    }
}
```

Лістинг 2.1. Модель документа *“Секція”*

Рис. 3.4 Модель документа «Секція»

Завдання

"Завдання" ("Issues") є необхідним документом для розподілу завдань між лікарями. Кожен лікарь може призначити певне "завдання" іншому лікарю, якщо тільки цей лікарь не вище його в правах доступу. "Завдання" також може бути призначена і самому собі, наприклад, потрібно, щоб певна діяльність була зафіксована і її, на даний момент, може виконати тільки сам лікарь, тоді він просто призначає сам собі "завдання".

Всі завдання зберігаються в окремій колекції. У тілі завдання зберігається посилання на документ лікаря, якому вона призначена, але в тілі документа "Лікарь" не зберігається посилання на завдання, які призначені йому. Даний підхід вирішує проблему освіти помилкових зв'язків між даними документами і уникнути можливих помилок в майбутньому. Для найкращого розуміння переваги такого методу, і що таке помилкові зв'язки, розглянемо приклад:

"Лікарь" знає список всіх завдань, призначених йому, а кожен документ "Завдання" знає кому воно призначене. При кожному перепризначенні завдання, потрібно буде також видаляти і посилання на перепризначення завдання з документа "Лікарь", тому вирішено зберігати зв'язок між лікарями і завданнями, призначеними їм, в односторонньому порядку, оскільки це зменшує кількість зв'язків між документами і прибирає ймовірність знаходження помилкових посилань.

Структура документа "Завдання" приведена в лістингу 2.2.

```
        {
    _id: <Issue-ID>, // ідентифікатор завдання
research: <Research-ID>,// ідентифікатор дослідження в якому створена завдання
from: <Researcher-ID>,// ідентифікатор того, ким створена
завдання
to: <Researcher-ID>,// ідентифікатор того, кому призначена
задача
 name: <Issuename>,  // назва завдання
desc: <Issuedescription>,// опис завдання
date_start: <Dateofissue>,    // дата початку виконання / створення
завдання
deadline: <Datawhenshouldbeend>,// дата, коли задача повинна бути
завершена date_closed: <Datewhenissuereallyend>,// дата, реального
завершення задачі priority: <Priority-level>,// приоритет задачі
closed: <bool>,          // чи закрита задача
db_section: {// секція з прапорами бази даних
        _id: <Section-type-ID>,
sec_data:{
deleted:
<Bool>,
date_created: <date>,
date_modified: <date>
        }
    }
}
```

Рис. 3.5 Модель документа «Задача»

Ролі

Розмежування прав проводиться по засобом привласнення, кожному лікарю, різних ролей. "Роль" являє собою документ, в якому зберігається набір прав доступу, присвоєний їй. Структура "Ролі" приведена в лістингу 2.3.

```
{
        _id : <Role-ID>,
priority : <Int>
        privileged : <BOOL>,
individuality : <BOOL>,
        read_only : <BOOL>,
        append : <BOOL>,
}
```

Лістинг 2.3. Модель документа *“Роль”*

Рис. 3.6 Модель документа «Роль»

Кожна роль має наступний набір полів:

● _id- ідентифікатор ролі. Задається явно, при додаванні ролі.

● priority - пріоритетність ролі. Потрібна, щоб розмежувати права розподілу завдань.

● privileged - логічне поле, вказує на те, чи володіє лікарь привілейованим статусом. YES - і має і відповідно має доступ до колекції (або БД) всіх випробовуваних), NO - немає, тільки через дослідження.

● individuality - логічне поле, вказує на те, в знеособленому чи вигляді, лікарю, що має дану роль, надаються дані пацієнтів. YES - дані надаються без змін, NO - в знеособленому вигляді.

● read_only - логічне поле, вказує на те, чи має право лікарь, який має цю роль, вносити зміни в уже занесені дані піддослідних.

● append - логічне поле, вказує на те, чи має право лікарь, який має цю роль, додавати нові дані до документів пацієнтів.

У документі "Лікарь", зберігається не сам документ "Роль", а посилання на даний документ. Самі ж "Ролі" зберігаються в окремій колекції. Даний підхід дозволяє, в разі чого, легко змінювати права доступу цілої групи лікарів, що мають певну роль. При змінюванні або додаванні будь-яких даних до певної ролі, потрібно буде провести зміни лише в одному документі, а не будувати складні запити для створення даної зміни в кожному документі "Лікарь".

Лістинги основних об'єктів

```
          {
 _id : <Researcher-ID>,
  rer_main_section : {
_id : <Section-type-ID>,                    // ідентифікатор лікаря, генерується випадково
      sec_data : {                          // секція з основними даними




last_name             : <'Прізвище'>,
first_name third_name : <'Ім'я'>,
age                   : <'По-батькові'>,
                      : <Bool>,
                      : <'Вік'>,


                                  // true - чоловічий, false - жіночий



            dBirth: <'Дата народження'>
            job: <'Місце роботи'>
            }




},
rer_resources_section : {
_id : <Section-type-ID>,                        // ресурсна секція
sec_data : {
equipment : [
{



                                  // масив обладнання



  eq_name : <'Назва обладнання'>,
  eq_desc : <'Опис обладнання'>
  },
  ...,
  {
  eq_name : <'Назва обладнання'>,
  eq_desc : <'Опис обладнання'>
  },
  ],
  methods : [//масив дослідницьких методів
  {
  md_name : <'Назва методу'>,
```

```
md_desc : <'Опис методу'>
},
...,
{
md_name : <'Назва методу'>,
md_desc : <'Опис методу'>
},
],
}
},
rer_busyness_section : {
_id : <Section-type-ID>,
sec_data : {
```

```
// бізнес секція
```

```
status  : <'Поточний статус (зайнятий; вільний ...'>,
role : <Role-ID>
}
},
rer_db_section : {// секція з прапорами бази даних
_id : <Section-type-ID>,
sec_data : {
login : "Login",// Логін лікаря
pass : "Password hash",// Хешкод пароля для входу в систему
deleted : <Bool>,
date_created : <date>,
date_modified  : <date>
}
}
}
```

Лістинг 2.4. Модель документа “Лікарь”

Рис. 3.7 Модель документа «Лікарь»

```
{
_id : <TestSubject-ID>,
ts_main_section : {
_id : <Section-type-ID>,
                                        // ідентифікатор пацієнта, генерується випадково
                                        // секція з основними даними

sec_data :                  {
last_name first_name third_name Прізвище
age                         : <'Ім’я'>,
                            : <'По-батькові'>,
                            : <Bool>,
                            : <'Стать'>,
                            : <'Вік'>,

                                    // якщо значення true, то вищеописані поля порожні і
                                    // не розглядаються

      dBirth: <'Дата народження'>
      }
      },
      ts_0_section : {// призначена для користувача секція
```

```
_id : <Section-type-ID>,
sec_data :  {
data1    :
<Some-
Data>,
data2    :
<Some-
Data>,
data3    :
<Some-
Data>,
            ...
dataN : <Some-Data>,
        }
    },
    ...
ts_N_section : {
_id : <Section-type-ID>,
sec_data :  {
       data1 : <Some-Data>, data2 : <Some-Data>, data3 : <Some-Data>,
            ...
dataN : <Some-Data>,
        }
    },
db_section : {// секція з прапорами бази даних
_id : <Section-type-ID>,
sec_data :  {
deleted       :
<Bool>,
date_created  :
<date>,
date_modified
: <date>
        }
    }
```

Лістинг 2.5. Модель документа *“Пацієнт”*

Рис. 3.8 Модель документа «Пацієнт»

Висновки

У цьому розділі розроблено інформаційну модель для зберігання медичних карт. Запропоновано оптимальний вибір моделі не реляційної бази даних – документо-орієнтованої бази даних. На основі вибраної моделі, запропоновано обрати платформою для зберігання медичних даних MongoDB.

Приведено та описано структуру бази даних для медичних карт в порядку розгортання від основних документів до побічних.

ЗАГАЛЬНІ ВИСНОВКИ ПО РОБОТІ

З огляду на конкретні проблеми зберігання, поставлені Великими даними, такі як масштабованість, доступність, інтеграція та безпека, традиційні системи вважаються нездатними обробляти існуючий сценарій даних.

NoSQL виявився життєздатним рішенням проблеми Великих даних. Особливості NoSQL, такі як динамічна схема, автоматичне шардування, автоматична реплікація та можливості інтегрованого кешування пом'якшують труднощі, які ставлять великі дані, перед традиційними системами.

Різні дослідницькі роботи класифікували доступні рішення NoSQL на основі підтримуваних моделей даних. Однак найбільш прийнята класифікація класифікує рішення NoSQL на основі з чотирьох моделей даних, а саме орієнтованих на документ, графові, ключ-значення та стовпчикові. Було досліджено переваги та недоліки кожної з цих моделей даних, щоб забезпечити аналіз випадків використання, які найбільше підходять для кожної моделі даних.

Це рішення ґрунтується на багатьох факторах, які можуть мати технічний та нетехнічний характер. Ця робота забезпечує аналіз особливостей 80 рішень NoSQL для полегшення прийняття рішень з цього приводу.

Наведено класифікацію наявних форматів файлів великих даних на п’ять категорій, а саме послуги на основі тексту, рядків, стовпців, пам’яті та зберігання даних. Формати на основі тексту втратили корисність в еру великих даних. Однак використання формату JSON та XML є загальним, враховуючи той факт, що вони легкі та зручні для читання.

Hadoop забезпечує кращі способи зберігання та форматування даних у файлах. Найбільш близьким до реляційного способу зберігання даних є

формат на основі рядків. SequenceFile та Avro належать до цієї категорії, серед яких перша є найбільш примітивною на основі рядків формат, що надається користувачам. Також доступні інші спеціалізовані формати на основі рядків, які використовують SequenceFile в своїй основі в Хадупі. До них належать MapFile, SetFile та BloomMapFile. Avro - це найбільш часто використовуваний формат рядків і є кращим для програм, які можуть вимагати всіх або більшості стовпців.

У більшості сценаріїв великих даних, як відомо, формати файлів на основі стовпців працюють краще, ніж на основі рядків аналоги, оскільки більшість запитів вимагають отримання кількох стовпців для декількох рядків.

У роботі розроблена інформаційна модель для медичних карт та обрано оптимальну платформу Великих даних MongoDB для зберігання медичних даних, на основі вибору документо-орієнтованої бази даних. Вибір саме документо-орієнтованої СУБД заснований на тому, що на відміну від реляційних аналогів тут не накладаються обмеження на набір полів у документа MongoDB задовольняє всі вимоги. Також це документо-орієнтована СУБД. Її було обрано саме через популярність, схожість підходу до зберігання даних з реляційними СУБД та формат зберігання даних BSON.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Big Data Related Technologies, Challenges and Future Prospects, 2014. – 89 с.

2. Інтернет великих даних [Електронний ресурс] – Режим доступу до ресурсу: https://www.bizmaster.xyz/2020/12/internet-velykyh-danyh.html.

3. Вся статистика интернета и соцсетей на 2021 год [Електронний ресурс] – Режим доступу до ресурсу: https://www.web-canape.ru/business/vsya-statistika-interneta-i-socsetej-na-2021-god-cifry-i-trendy-v-mire-i-v-rossii/.

4. SQL schema design: foundations, normal forms, and normalization. Information Systems / H. Köhler, S. Link. – 2018

5. Real-time urban microclimate analysis using internet of things. // Internet of Things Journal. – 2018. – С. 500–511.

6. An Artificial Intelligence Enabled Data Analytics Platform for Digital Advertisement. / N. Albayrak, A. Özdemir, E. Zeydan. // In 2019 22nd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN). – 2019. – С. 239–241.

7. The big data system, components, tools, and technologies: a survey. / T. R.Rao, P. Mitra, R. Bhatt, A. Goswami. // Knowledge and Information Systems. – 2018. – С. 1–81.

8. Big Data Storage for a Health Predictive System. / J. R. Cordeiro, O. Postolache. // In 2018 International Symposium in Sensing and Instrumentation in IoT Era (ISSI). – 2018. – С. 1–6.

9. U.S. Patent Application / J.Zhao, M. Lai, H. Tian, H. Chang. // U.S. Patent Application. – 2019. – №10. – С. 673.

10. Effects of storage heterogeneity in distributed cache systems. / K. S. Reddy, S. Moharir, N. Karamchandani. // In 2018 16th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt). IEEE. – 2018. – С. 1–8.

11. NoSQL databases. Lecture Notes / C. Strauch, U. S. Sites, W. Kriha. // Stuttgart Media University. – 2011. – №20.

12. NoSQL Database Classification: New Era of Databases for Big Data. / [B. Acharya, A. K. Jena, J. M. Chatterjee та ін.]. // International Journal of Knowledge-Based Organizations (IJKBO). – 2019. – №9. – С. 50–65.

13. A performed load balancing algorithm for public Cloud computing using ant colony optimization. / A. Ragmani, A. El Omri, N. Abghour. // Recent Patents on Computer Science. – 2018. – №11. – С. 179–195.

14. Zheng, X. (2018). Database as a Service-Current Issues and Its Future. arXiv preprint arXiv:1804.00465

15. NoSQL Databases / C. Strauch. // Lecture, Stuttgrat Media University.

16. Scalable SQL and NoSQL data stores. / Cattell. // Acm Sigmod Record. – 2011. – №39. – С. 12–27.

17. Will NoSQL databases live up to their promise? / Leavitt. // Computer. – 2010. – №43. – С. 12–14.

18. NoSQL–Death to Relational Databases. CodeMash Presentation / B. Scofield. – 2010.

19. Data modeling for analytical queries on document-oriented DBMS. // In Proceedings of the 33rd Annual ACM Symposium on Applied Computing. – 2018. – С. 541–548.

20. Comparison of NoSQL Database and Traditional Database-An emphatic analysis / S. Kumar. // JOIV: International Journal on Informatics Visualization. – 2018. – №2. – С. 51–55.

21. Managing big RDF data in clouds: Challenges, opportunities, and solutions. Sustainable Cities and Society / Elzein, Majid, Hashem, Yaqoob. – 2018. – №39. – С. 375–386.

22. An introduction to Graph Data Management. In Graph Data Management / R. Angles, C. Gutierrez. – 2018. – С. 32.

23. Data Modeling Across the Evolution of Database Technology. // A Comprehensive Guide Through the Italian Database Research Over the Last 25 Years. – 2018. – C. 221–234.

24. Sstudy of NoSQL Database for enterprises. // 2018 International Symposium on Computer, Consumer and Control (IS3C). – 2018. – C. 436–440.

25. Miller, A. H., Fisch, A. J., Dodge, J. D., Karimi, A. H., Bordes, A., & Weston, J. E. (2018). U.S. Patent Application No. 16/002,463

26. Nguyen, P. A. P., Kryze, D., Vassilakis, T., & Lerios, A. (2019). U.S. Patent Application No. 10/176,236

27. CAP Theorem: Revision of Its Related Consistency Models. // The Computer Journal. – 2019.

28. Building Tunable CRDTs : дис. докт. / Rijo., 2018

29. Tendencies of technologies and platforms in smart cities: A state-of-the-art review / Chamoso, González-Briones, Rodríguez, Corchado. // Wireless Communications and Mobile Computing. – №2018.

30. Carroll, J. R., & Robertson, F. R. (2019). A COMPARISON OF PHASOR COMMUNICATIONS PROTOCOLS (No. PNNL-28499). Pacific Northwest National Lab.(PNNL), Richland, WA (United States)

31. Measuring the Performance of Data Placement Structures for MapReduce-based Data Warehousing Systems. // International Journal of New Computer Architectures and Their Applications. – 2018. – C. 11–21.

32. Performance Analysis of RDBMS and Hadoop Components with Their File Formats for the Development of Recommender Systems. // 2018 3rd International Conference for Convergence in Technology (I2CT). – 2018.

33. ACM Efficient Analytics on Encrypted Data. // Proceedings of the 11th ACM International Systems and Storage Conference. – 2018. – C. 112–121.

34. Open Data Standards for Administrative Data Processing / R. White. – 2018.

35. Next-Generation Big Data: A Practical Guide to Apache Kudu, Impala, and Spark. Apress / Quinto. – 2018.

36. Handling Small Size Files in Hadoop: Challenges, Opportunities, and Review. // In Soft Computing in Data Analytics. – 2019. – С. 653–663.

37. Novel BI data architectures. // International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). – 2018. – №41.

38. Boosting data filtering on columnar encoding with SIMD / H. Jiang, A. J. Elmore. // Proceedings of the 14th International Workshop on Data Management on New Hardware. – С. 6.

39. Supporting Columnar In-memory Formats on FPGA: The Hardware Design of Fletcher for Apache Arrow. // International Symposium on Applied Reconfigurable Computing. – 2019. – С. 32–47.

40. A proposed solution and future direction for blockchain-based heterogeneous medicare data in cloud environment / Kaur, Alam, Chang та ін.]. // Journal of medical systems. – 2018. – №42. – С. 156.

41. A blockchain-based framework for data sharing with finegrained access control in decentralized storage systems / S. Wang, Y. Zhang, Y. Zhang. – 2018. – №6. – С. 38437–38450.

42. Cloud-based big data analytics—a survey of current research and future directions / S. Khan, K. A. Shakil, M. Alam. // Big Data Analytics. – 2018. – С. 595–604.

43. Photogroup: Decentralized Web Application Using Ethereum Blockchain / Paralkar, Yadav, Kumari, Kulkarni. // International Research Journal of Engineering and Technology. – 2018. – №5. – С. 489–492.

44. A Survey and Evaluation of the Potentials of Distributed Ledger Technology for Peer-to-Peer Transactive Energy Exchanges in Local Energy Markets / Siano, De Marco, Rolán, Loia. // IEEE Systems Journal. – 2019.

45. Biometric Encryption in Cloud Computing: A Systematic Review. / M. Ansar, M. Fatima. // International Journal of Computer Science and Network Solutions. – 2018. – №6. – С. 10–19.

46. A distributed cooperative control framework for synchronized reconnection of a multi-bus microgrid / D. Shi, X. Chen, X. Zhang. // IEEE Transactions on Smart Grid. – 2018. – №9. – С. 6646–6655.

47. Blockchain meets IoT: An architecture for scalable access management in IoT / O. Novo. // IEEE Internet of Things Journal. – 2018. – №5. – С. 1184–1195.

48. Data modeling for analytical queries on document-oriented DBMS / Soransso, Cavalcanti. // Proceedings of the 33rd Annual ACM Symposium on Applied Computing. – 2018. – С. 541–548.

49. Sun, J., & Milani, C. V. B. (2018). U.S. Patent Application No. 15/406,643

50. Geissinger, S. (2018). U.S. Patent Application No. 15/354,921

51. The Little MongoDB Book [Електронний ресурс] – Режим доступу до ресурсу: https://www.pvsm.ru/download/mongodb-ru.pdf.