ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

Пояснювальна записка

до магістерської роботи

на ступінь вищої освіти магістр

на тему: «Google BigQuery як альтернатива MySQL в контексті мікросервісної архітектури»

Виконав: студент 6 курсу, групи ППЗМ–61

спеціальності 121 Інженерія програмного забезпечення

(шифр і назва спеціальності/спеціалізації)

Трофименко В.В.

(прізвище та ініціали)

Керівник Жебка В.В.

(прізвище та ініціали)

Рецензент _____

(прізвище та ініціали)

Нормоконтроль

(прізвище та ініціали

Київ – 2022

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти -«Магістр»

Спеціальність підготовки – 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

Інженерії програмного забезпечення

Негоденко О.В.

“____” _____2022 року

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

ТРОФИМЕНКО ВАДИМУ ВОЛОДИМИРОВИЧУ

(прізвище, ім’я, по батькові)

1. Тема роботи: Google BigQuery як альтернатива MySQL в контексті мікросервісної архітектури

Керівник роботи: Жебка Вікторія Вікторівна, д.т.н., професор кафедри ІПЗ

(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом вищого навчального закладу від «__»__2022 року №__.

2. Строк подання студентом роботи

3. Вхідні дані до роботи

Офіційна документація MySQL;

Офіційна документація Google BigQuery;

Науково-технічна література з питань, пов’язаних з базами даних;

4. Зміст розрахунково-пояснювальної записки(перелік питань, які потрібно розробити)

4.1. Опис та аналіз загальної характеристики систем управління баз даних

4.2. Опис математичної системи динамічної бази даних

4.3. Порівняльний аналіз функціональних можливостей MySQL та Google BigQuery

5. Перелік демонстраційного матеріалу (назва основних слайдів)

6. Дата видачі завдання_____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів робот	Примітка
1	Підбір науково-технічної літератури	25.09-21.10	
2	Опис та аналіз загальної характеристики систем управління баз даних	25.10-10.11	
3	Опис математичної системи динамічної бази даних	15.11-15.12	
4	Порівняльний аналіз функціональних можливостей MySQL та Google BigQuery	20.12-27.01	
5	Вступ, висновки, реферат	01.02-14.02	
6	Розробка презентації	17.02-01.03	

Студент _____

Керівник роботи _____

ЗМІСТ

Скорочення……...**3**

Реферат……...**4**

ВСТУП…….. **5**

1. ЗАГАЛЬНА ХАРАКТЕРИСТИКА СИСТЕМ УПРАВЛІННЯ БАЗ ДАНИХ……...**8**

1.1. Моделі зберігання даних……... 8

1.2. Математичний опис динамічної моделі бази даних……. 13

1.3. Реляційні бази даних та визначення їх загальної архітектури…….16

1.4. Тенденції розвитку сучасних СУБД……...22

2. ТЕХНІКО–ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ……**24**

2.1. Аналіз функціонального інструментарію MySQL……24

2.2. Особливості використання Google BigQuery…… 39

3. ОЦІНКА МОЖЛИВОСТЕЙ GOOGLE BIGQUERY ТА MYSQL…**54**

3.1. Порівняння переваг та недоліків досліджуваних баз даних…… 54

3.2. Перспектива використання Google BigQuery…… 58

ВИСНОВКИ……... **63**

ПЕРЕЛІК ПОСИЛАНЬ……**64**

ДОДАТКИ…….…..**69**

Скорочення

БВФ	Багатомірні вагові функції
БД	База даних
ЕОМ	Електронно-обчислювальна машина
ІПР	Імовірності правильного розпізнавання
ІТ	Інформаційні технології
Ксп	Колоночний співпроцесор
ОК	Об’єкти контролю
СУБД	Система управління базами даних
ТПО	Таблиці попередніх обчислень
ACID	(Atomicity, Consistency, Isolation, Durability) Атомарність-Послідовність-Ізоляція-Довговічність
IAM	Identity and Access Management
ODL	Object Definition Language
ODMG	Object Data Management Group
ORC	Optimized Row Columnar
XML	eXtensible Markup Language

РЕФЕРАТ

Система управління базами даних – це програмний інструмент або іншими словами інтерфейс між кінцевим користувачем та програмою, і природно, самої базою даних, на якій виконуються завдання. За допомогою СУБД легше і зручніше працювати з інформацією. Наприклад, створювати, оновлювати, шукати, видаляти і відновлювати дані в базах даних, а також визначати взаємозв'язки між її компонентами (таблицями для реляційного типу).

Мета – покращення процесу аналізу та вивчення даних за допомогою Google BigQuery як альтернативи MySQL.

Об’єкт дослідження – процес аналізу та вивчення даних.

Предметом дослідження є методи та засоби роботи з Google BigQuery та MySQL.

Розглянуто існуючі на даний час моделі зберігання даних та сказано, що найбільш розповсюдженими моделями є постреляційна, багатомірна, об'єктно-орієнтована. Для кожної моделі дано короткий опис та дано схематичне зображення. Для динамічної моделі бази даних зроблено математичний опис і для реляційних баз дано поглиблене визначення та опис. Встановлено, що реляційні бази даних надійні, а транзакції завжди виконуються, як очікується, завдяки принципам ACID. Наголошено, що Перспективи розвитку архітектур СКБД пов'язані з розвитком концепції обробки нетрадиційних даних та їх інтеграцією, обміном даними з різних СКБД, багатокористувацької технології в локальних мережах. Проведено аналіз функціонального інструментарію MySQL та Google BigQuery. MySQL є рішенням для малих і середніх додатків, а Google BigQuery використовується для великих хмарних баз даних. Для досліджуваних систем дано їх математичний опис та наведено схематичну архітектуру, вказано переваги та недоліки, наголошено на областях практичного використання.

Зроблено порівняння досліджуваних систем та вказано можливий шлях імпорту даних із MySQL до Google BigQuery. Зроблено висновок про те, що можливості Google BigQuery можна розширити за допомогою ряду сторонніх інструментів. Наприклад, інтегрувавши його з Google Таблиці, Microsoft Excel, QlikView, BIME Analytics та Microsoft Power BI.

Встановлено, що перспективність використання Google BigQuery полягає у розширенні можливостей сумісного використання даної бази даних з іншими програмними продуктами та оптимізація продуктивності запитів.

Ключові слова: *MySQL та Google BigQuery, бази данних, хмарні дані, принципи ACID, реляційні бази даних.*

ВСТУП

Більшість даних у сучасних системах зберігається саме у базах даних. Бази даних залишаються невід'ємною частиною кожної комп'ютерної системи. З переходом значної кількості програмних додатків у он-лайн, деякі архітектурні особливості баз даних втратили свою актуальність.

Актуальність дослідження. Автоматизація процесу створення програмного продукту набула великих змін з появою хмарних сервісів та серверів, а також популяризації мікросервісної архітектури програмних додатків. СУБД є основним елементом будь-якої програми, вони зберігають та обробляють усі важливі і другорядні дані, необхідні для функціонування програм та підтримки користувацького інтерфейсу. Завжди є хоча б одна база даних, або в деяких випадках безліч баз даних, особливо при зберіганні великих даних, які працюють у фоновому режимі, синхронно, щоб підтримувати додатки функціональними. Бази даних можна вважати сховищами інформації, де вона пов'язана між собою, сортується та зберігається структурно або напівструктурно, що робить її легкою для пошуку та доступною у використанні [5].

Система управління базами даних (СУБД) – це програмний інструмент або іншими словами інтерфейс між кінцевим користувачем та програмою, і природно, самою базою даних, на якій виконуються завдання. За допомогою СУБД легше і зручніше працювати з інформацією. Наприклад, створювати, оновлювати, шукати, видаляти і відновлювати дані в базах даних (БД), а також визначати взаємозв'язки між її компонентами (таблицями для реляційного типу). Сьогодні, при розробці більш-менш серйозних веб-додатків або прикладних рішень гостро постає питання у виборі системи управління базами даних для виконання покладених на неї завдань. База даних повинна відповідати ряду вимог, серед яких: надійність, розширюваність, продуктивність і можливість витримувати великі навантаження [11]. У різних областях діяльності накопичено величезну кількість даних, що веде до посилення вимог до їх обробки та зберігання, зокрема до продуктивності

систем управління базами даними. Дана проблема особливо актуальна для даних, що вимагають глибокого аналізу. У зв'язку з цією ситуацією з'являються нові підходи до побудови таких систем, які повинні подолати недоліки існуючих. Представлене дослідження присвячене порівнянню можливостей Google BigQuery та MySQL в контексті мікросервісної архітектури.

Мета – покращення процесу аналізу та вивчення даних за допомогою Google BigQuery як альтернативи MySQL.

Об’єкт дослідження – процес аналізу та вивчення даних.

Предметом дослідження є методи та засоби роботи з Google BigQuery та MySQL.

Завдання дослідження полягають в наступному:

- розглянути основні моделі зберігання даних та зробити математичний опис динамічної моделі бази даних;
- перелічити особливості реляційних баз даних та дати визначення їх загальної архітектури;
- оцінити тенденції розвитку сучасних СУБД;
- дослідити функціональний інструментарій MySQL та Google BigQuery;
- провести порівняння переваг та недоліків досліджуваних баз даних;
- розглянути перспективи використання Google BigQuery.

Методи дослідження. У процесі досліджень використовувались: методи фільтрації, математичне моделювання і прогнозування, комп’ютерне моделювання для аналізу та перевірки отриманих теоретичних положень, методи порівняння, методи проєктування.

У роботі для вирішення поставлених завдань були використані спеціальні і загальнонаукові методи дослідження. Способи синтезу та аналізу. Вирішення поставлених у роботі завдань здійснювалося з використанням системного підходу в доборі матеріалу, методів індуктивного і логічного аналізу, спостереження та статистичні методи аналізу літературних даних.

Наукова новизна отриманих результатів.

На основі виконаних теоретичних і експериментальних досліджень вирішено прикладну проблему розробки теоретичних та практичних засад для обробки баз даних, а саме:

- приведено класифікацію методів для підвищення швидкодії роботи з великими обсягами даних;
- подальшого розвитку отримав метод паралельної організації запитів великих баз даних;
- запропоновано метод адаптивної оптимізації ресурсів при виконанні паралельних запитів великих баз даних.

Новизна дослідження. Зроблено широкий літературний пошук з детальним аналізом наукової інформації. Проведено систематизацію та адаптацію отриманих літературних результатів. Встановлено рекомендації з використання отриманих даних.

Структура та обсяг роботи. Відповідно до мети і завдань дослідження структура роботи складається зі вступу, трьох розділів, висновків та списку використаної літератури. За час роботи опрацьовано 52 літературних джерела. Зміст роботи викладено на 70 сторінках машинописного тексту.

Джерелами інформації для вирішення перерахованих вище завдань є збірники наукових праць, монографії, періодична література, підручники та довідники, періодичні фахові журнали.

1. ЗАГАЛЬНА ХАРАКТЕРИСТИКА СИСТЕМ УПРАВЛІННЯ БАЗ ДАНИХ

1.1. Моделі зберігання даних

Збережені в БД записі мають логічну структуру – модель представлення даних. Найбільш розповсюдженими моделями раніше були ієрархічна, мережна, реляційна. В даний час розроблені нові моделі: постреляційна, багатомірна, об'єктно-орієнтована. На їхній основі створюються комбіновані моделі: об’єктно-реляційна, дедуктивне-об’єктно-орієнтовані, семантичні, концептуальні [14].

У деяких СУБД підтримується одночасно кілька моделей даних (Cache). Перш, ніж перейти до вивчення реляційних систем БД, необхідно ознайомиться з до реляційними СУБД так як внутрішня організація реляційних систем багато в чому заснована на використанні методів ранніх систем, це буде корисно для поняття шляхів розвитку постреляційних СУБД. Обмежуємося розглядом тільки загальних підходів до організації трьох типів ранніх систем:

- систем, заснованих на інвертованих списках;
- ієрархічних [18];
- мережних систем керування базами даних;
- об’єктно-орієнтовані базами даних.

Почнемо розгляд досліджуваної проблематики з деяких найбільш загальних характеристик ранніх систем. Ці системи активно використовувалися протягом багатьох літ, довше, ніж використовується яка-небудь з реляційних СУБД. Насправді деякі з ранніх систем використовуються навіть у наш час, накопичені величезні бази даних, і однієї з актуальних проблем інформаційних систем є використання цих систем разом із сучасними системами. Усі ранні системи не ґрунтувалися на яких-небудь абстрактних моделях.

Найбільш поширеним є підхід до поділу моделей баз даних на сім типів : 1) ієрархічні; 2) мережеві; 3) реляційні; 4) лінійно-рекурентні; 5) об’єктно-орієнтовані; 6) об’єктно-реляційні; 7) напівструктуровані.

Ієрархічна структура даних – це структура, де будь-який об’єкт може підпорядковуватися лише одному об’єкту вищого рівня, а йому багато об’єктів нижчого рівня [32]. Такий зв’язок між даними називають «один до багатьох». За ієрархічним принципом побудовані файлові структури даних на дисках персонального комп’ютера. На рисунку 1.1. показані зв’язки ієрархічної моделі БД.

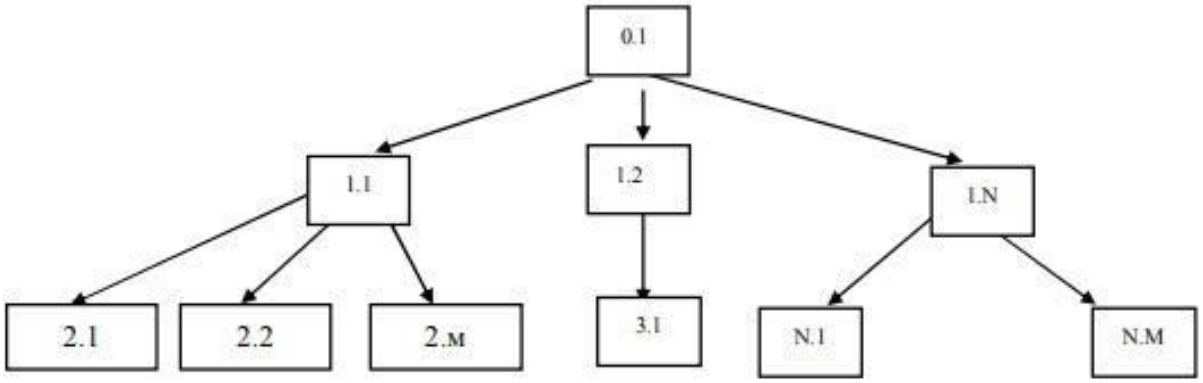


Рисунок 1.1. Ієрархічна модель БД

Мережева структура БД характеризується тим, що будь-який об’єкт одного рівня (одної групи даних) може мати довільні зв’язки з об’єктами іншого рівня. Такі зв’язки називаються «багато до багатьох». Мережеві структури можна описати у вигляді таблиці, де у першому горизонтальному рядку записують об’єкти одного рівня, а у першому вертикальному – іншого. Така таблиця добре ілюструє зв’язки між об’єктами, але може мати багато порожніх елементів, що призводить до значної надлишковості БД [31]. На рисунку 1.2. показані зв’язки мережевої моделі БД.

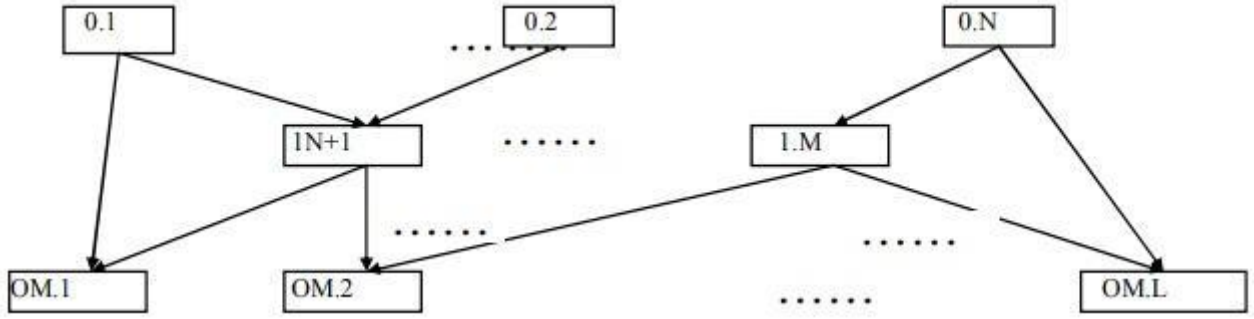


Рисунок 1.2. Схематичне зображення мережевої моделі БД

Реляційна БД є найпоширенішим типом і характеризується поданням даних у вигляді декількох таблиць і зв'язаними між собою кортежів та атрибутів. Один з найпростіших типів зв'язків є «один до одного». Потрібні дані будуть черпатися з двох таблиць. Отже, для зв'язку між таблицями використовують поле, значення якого не повторюється в різних записах. Це поле називається ключовим. Якщо реляційні таблиці мають спільні поля, то зміни у спільному полі в одній таблиці автоматично відображаються у всіх таблицях. Мета запровадження реляційних зв'язків – мінімізувати дублювання даних і забезпечити можливість опрацювати (шукати) дані з декількох таблиць, що забезпечує значне зменшення надлишковості БД [12]. На рисунку 1.3 показана структура реляційної моделі БД.

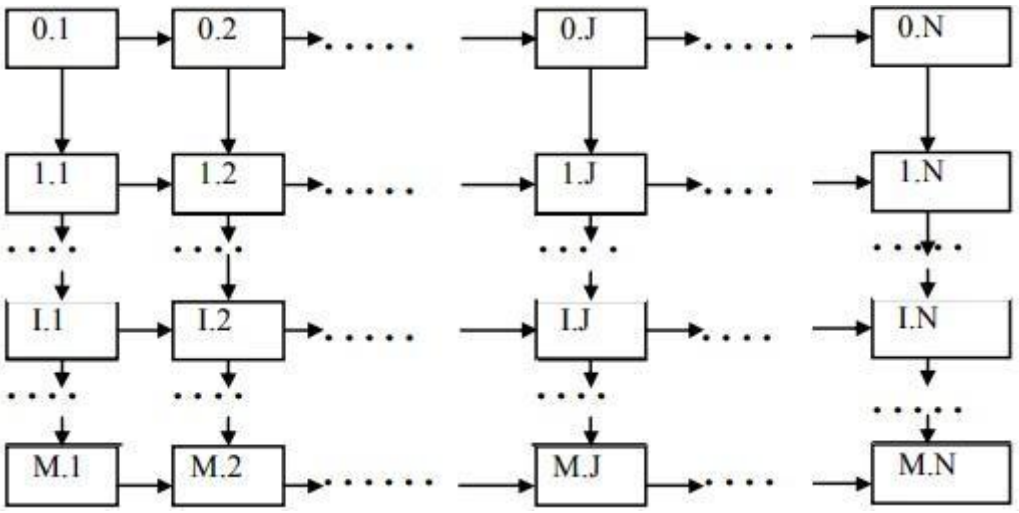


Рисунок 1.3. Схематичне зображення реляційної моделі

Лінійно-рекурентні БД Лінійно-рекурентні БД є новим типом структурної організації баз даних, яка формується на основі рекурентних властивостей

теоретико-числового базису Галуа. Важливою характеристикою таких БД є максимальна компактність адресації даних, в тому числі при багаторівневій архітектурі. Позитивною характеристикою лінійно-рекурентних БД є можливості ефективного захисту від помилок та несанкціонованого доступу [4].

Об’єктно–орієнтовані БД. Один із підходів до забезпечення сумісності систем БД з парадигмою об’єктно-орієнтованого проєктування зв’язаний з розширенням системи розумінь, на основі об’єктно-орієнтованих мов програмування, таких як C++ або Java. В традиційному програмуванні розуміється, що після завершення циклу роботи програми її об’єкти безповоротно втрачаються, в той час як принципові умови роботи будь-якої СУБД полягають у тому, що об’єкти повинні зберігатися необмежено довгий час, поки не будуть змінені або знищені примусово, як це відбувається у файлових системах. «Чисту» об’єктно-орієнтовану модель представлення даних, названу ODL (Object Definition Language) - мова визначення об’єктів, яка в свій час була стандартизована дослідницькою групою Object Data Management Group (ODMG) [1]. На рисунку 1.4. показана структура об’єктно-орієнтованої БД.

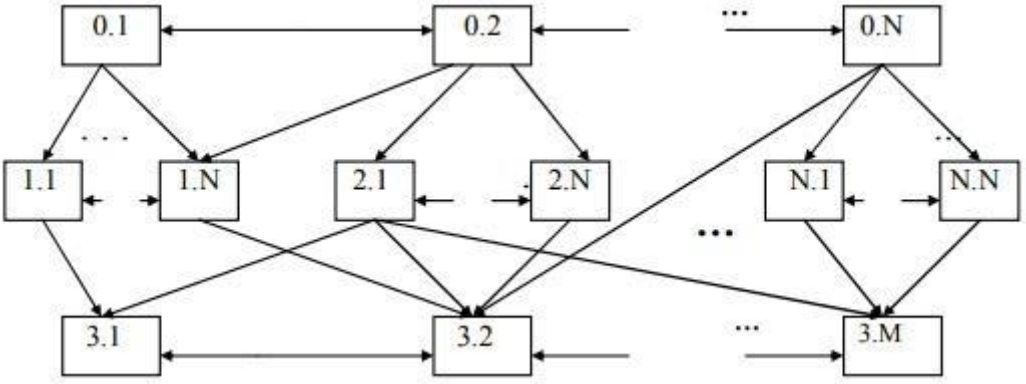


Рисунок 1.4. Об’єктно-орієнтовані БД

Об’єктно-реляційні БД. Ця модель є частиною самого останнього стандарту SQL, названого SQL-99 (або SQL-1999, також SQL3), і являє собою варіант розширення звичайної реляційної моделі за рахунок формалізації багатьох загальноприйнятих концепцій об’єктно-орієнтованого проєктування.

Вказаний стандарт служить основою для побудови об’єктно-реляційних систем БД, які сьогодні випускаються всіма основними постачальниками комерційних СУБД. Ці системи, однак, значно відрізняються в деталях практичної реалізації початкових концепцій [3]. На рисунку 1.5. показана структура об’єктно-реляційної БД.

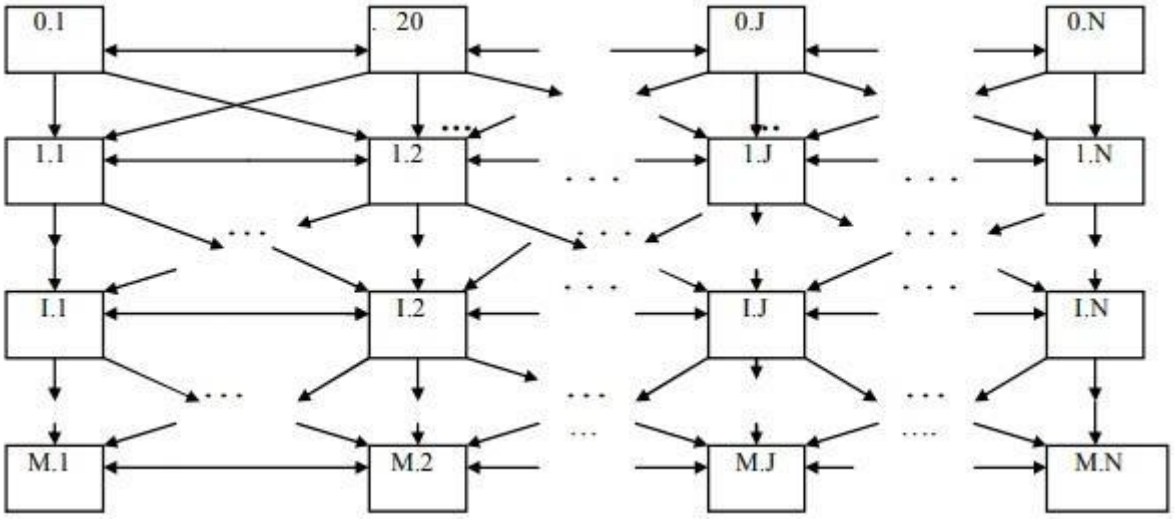


Рисунок 1.5. Об’єктно-реляційна модель БД

Напівструкторовані БД. Одне із останніх досягнень,призначених вирішити велику кількість актуальних проблем технологій СУБД, включаючи необхідність об’єднання традиційних баз даних з іншими джерелами інформації, такими, як Web-сторінки найрізноманітнішої структури. В той час, як об’єктно-орієнтовані або об’єктно-реляційні системи пропонують використання фіксованих схем для кожного класу або відношень, модель напівструктурованих даних відрізняється набагато більш високою гнучкістю представлення компонентів інформації [2].

Найбільш яскравим практичним втіленням моделі напівструктурованих даних є XML (eXtensible Markup Language - росширювана мова розмітки). XML-це специфікація опису «документів», що представляють собою набір елементів даних, функції яких визначаються відповідними тегами. Підтримка XML з часом стане найважливішим компонентом систем, що здійснюють

посередницькі функції між різнорідними джерелами даних, і навіть, що дуже можливо, послужить цілям гнучкого представлення інформації в БД [23].

Поняття моделі даних фактично узвичаїлося фахівців в області БД тільки разом з реляційним підходом. Абстрактні представлення ранніх систем з'явилися пізніше на основі аналізу і виявлення загальних ознак у різних конкретних систем. У ранніх системах доступ до БД вироблявся на рівні записів. Користувачі цих систем здійснювали явну навігацію в БД, використовуючи мови програмування, розширені функціями СУБД. Інтерактивний доступ до БД підтримувався тільки шляхом створення відповідних прикладних програм із власним інтерфейсом. Навігаційна природа ранніх систем і доступ до даних на рівні записів змушували користувача самого робити всю оптимізацію доступу до БД, без якої-небудь підтримки системи. Після появи реляційних систем більшість ранніх систем були оснащені «реляційними» інтерфейсами. Однак у більшості випадків це не зробило їх по-справжньому реляційними системами, оскільки залишалася можливість маніпулювати даними в природному для них режимі [21].

1.2. Математичний опис динамічної моделі бази даних

Основна мета кластерного аналізу – знаходження груп схожих об’єктів у вибірці. Спектр застосувань кластерного аналізу дуже широкий: його використовують в археології, антропології, медицині, психології, хімії, біології, державному управлінні, філології, маркетингу, соціології та інших дисциплінах. Однак, універсальність його застосування привела до появи великої кількості несумісних термінів, методів і підходів, що ускладнюють однозначне використання і несуперечливу інтерпретацію кластерного аналізу.

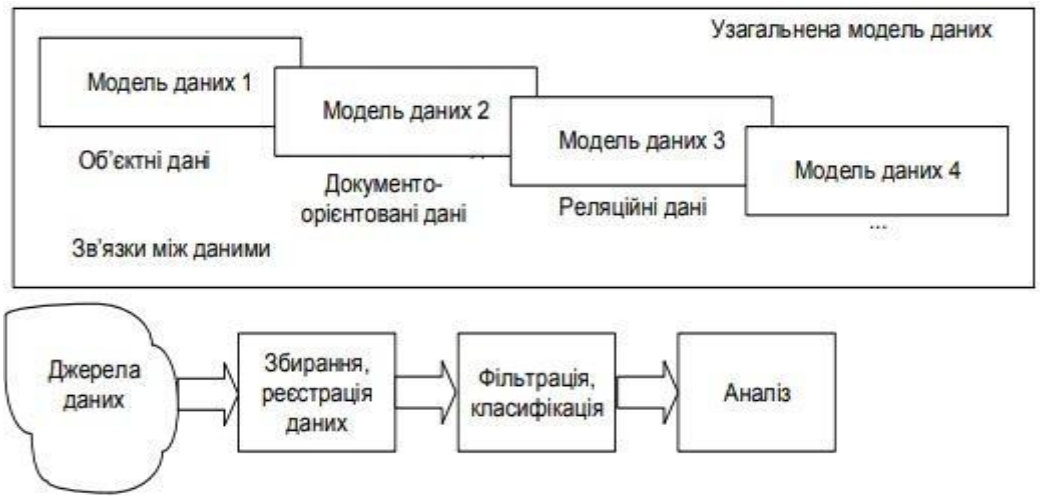


Рисунок 1.6. Алгоритм побудови діагностичних моделей нелінійних динамічних систем в умовах великих даних

Розробка моделі нелінійних динамічних систем для неструктурованих даних є вкрай складним завданням з наступних причин. По-перше, дані, як правило, представлені природною мовою (дані температури), що ускладнює роботу з ними. По-друге, повна відсутність визначеної структури накладає серйозні обмеження на можливі операції з даними. Автоматичне виділення структури в таких даних, як правило, не може бути виконано однозначним чином. В реляційних СУБД значення даних температури в основному представляються у вигляді рядкових констант. Формати цих констант в різних СУБД відрізняються один від одного. Крім того, способи запису даних температури змінюються в залежності від країни і це потрібно враховувати [22].

Аналізуючи математичну сторону представлення досліджуваних даних, нелінійні динамічні моделі можна представити у такому вигляді:

$$y_j(t) = \sum_{k=1}^{\infty} \sum_{i_1=1}^{\nu} \dots \sum_{i_k=1}^{\nu} \int_0^t \dots \int_0^t w^j{}_{i_1 i_2 \dots i_k}(\tau_1, \tau_2, \dots, \tau_k) \prod_{l=1}^{k} x_{i_l}(t-\tau_l) d\tau_l \qquad (1.1)$$

де $w^j_{i_1 \dots i_k}(\tau_1, \dots \tau_k)$ – багатомірні вагові функції (БВФ) k-го порядку по $i_1, \dots, i_k$ входам та j-му виходу (j=1,2,...,μ), ν, μ – кількість входів та виходів відповідно, x

(t) – вхідний вплив, y_j *(t)* – відгук об'єкта на *j*-му виході при нульових початкових умовах.

Обробка діагностичної інформації у вигляді набору даних являє собою досить важку та ресурсномістку задачу, що вимагає великих обсягів пам'яті ЕОМ та часу процесу діагностування. Тому доцільним є перехід від первинних даних – набору БВФ різних порядків – до простору діагностичних ознак. Для цього проводиться аналітичний огляд методів побудови класифікаторів об'єктів контролю (ОК) (статистичні методи, методи нечітких множин і нейронні мережі) у просторі діагностичних ознак. Встановлюється область ефективного використання статистичних методів: швидка побудова вирішуваних правил у вигляді поліномів малих порядків на основі репрезентативної навчальної вибірки [17].

Для оцінки якості системи класифікації в роботі застосовуються експериментальні показники: помилки класифікації, імовірності правильного розпізнавання (ІПР) і мінімум середнього ризику. Оцінка помилок класифікації визначається виразом:

$$\delta_i = L_i / N_i, \qquad (1.2)$$

де L_i – кількість об'єктів і-го класу, помилково віднесених до іншого класу k (k≠i); N_i _ кількість елементів і-го класу в вибірці; i=1, 2,..., m; m – кількість класів стану ОК.

Оцінка ІПР P, середня по всіх класах:

$$P = 1 - \sum_{i=1}^{m} L_i \cdot \left(\sum_{i=1}^{m} N_i \right)^{-1}, \qquad (1.3)$$

Оцінка середнього ризику R (середньої вартості прийняття рішення):

$$R = \sum_{i=1}^{m} \delta_i s_i p(\Omega_i) \qquad (1.4)$$

$$p(\Omega_i) = N_i / \sum_{j=1}^{m} N_j$$

де s_i – вартості помилок δ_i , – апріорна імовірність появи класу Ω_i.

Обґрунтовано необхідність створення інструментальних засобів діагностичного контролю, що одночасно забезпечують роботу запропонованих моделей ОК, формування простору діагностичних ознак і побудову ефективних класифікаторів в задачах багатоальтернативного розпізнавання станів ОК. В задачах модельної діагностики адекватність моделі реальним ОК треба розуміти не в розумінні точності опису відгуку об'єкта, а в розумінні її діагностичної цінності з погляду достовірного (надійного) розпізнавання технічного стану [8]. Тому при ідентифікації нелінійних динамічних ОК необхідно забезпечити в першу чергу високу точність оцінки перетинів БВФ малих порядків, що містять найбільшу кількість діагностичної інформації для побудови ефективної системи розпізнавання.

Розпізнавання станів ОК проводиться на основі їх опису в просторі вторинних діагностичних ознак, отриманих на основі діагностичних моделей:

$$\{w_k(t_1, t_2, \ldots, t_k)\}_{k=1,2,\ldots,K} \Rightarrow \boldsymbol{x} = (x_1, x_2, \ldots, x_n)'$$

(K – порядок БВФ, n – розмірність простору ознак, штрих – транспонування вектора). Пропонуються наступні способи стискування діагностичних моделей.

Елементи реалізації ідеї слабоструктурованої обробки й зберігання даних є в безсхемних БД, що відносяться до типу NoSQL (Not Only SQL, не тільки SQL) систем. Їхньою особливістю, зокрема, є горизонтальне масштабування сховища даних та підтримка пошуку й індексування по довільних полях, а в деяких БД є можливість формування будь-яких запитів вибірки даних. Найпростішим способом реалізації слабоструктурованого зберігання даних є динамічне сховище ключів і значень. Іншим підходом до забезпечення можливості динамічної зміни структури БД є стовпчикова реалізація зберігання даних (протилежно до рядкової в реляційних БД) [9].

1.3. Реляційні бази даних та визначення їх загальної архітектури

Реляційні бази даних мають на увазі збереження даних у таблицях, які пов’язані між собою кортежами даних. Кортежі поєднуються завдяки ключам: зовнішніми та основним. До реляційних баз даних застосовується як реляційна алгебра, так і зведення бази даних до нормальних форм, що значно оптимізує збереження даних, та обчислення у СУБД. Мова виконання запитів SQL дозволяє не тільки управляти даними всередині бази, а й проводити аналітику чи вибірку навіть із сирих даних. Більше того, ця мова дуже розповсюджена і підходить для виконання запитів на більшості РСУБД [10].

Реляційні моделі дуже прості, вони містять у собі лише базові концепти, тоді як зв’язки лише симулюються. Також у таблицях реляційних баз даних можна використовувати лише певні типи даних, що пропонуються. Недоліками реляційного типу СУБД можна вважати недостатню семантику моделювання і наявність складних типів та структур даних, які потрібно моделювати для збереження іноді базової інформації. Також у бази даних мала підтримка як самих типів даних, так і покращення таблиць для їх збереження. Часто виникають складні зв’язки між об’єктами, завдяки яким необхідно створювати додаткові таблиці з ключами («many-to-many»), що призводить до уповільнення роботи бази даних.

Реляційні моделі баз даних мають проблеми, які обумовлені їх структурою. По-перше, це використання моделі Атомарність-Послідовність-Ізоляція-Довговічність (ACID) кожним запитом, навіть коли це не потрібно [19]. По-друге, таку базу даних складно масштабувати, так як зв’язки між таблицями уповільнюють цей процес. Також недоліки помічаються у доступності та гнучкості такого виду СУБД. Традиційні методи масштабування мають на увазі розширення можливостей сервера бази даних, управління пам’яттю та кешуванням (запитів, відповідей), а також можливостей реплікації та зберігання у пам’яті запитів для їх моментального відтворення у випадку, коли це потрібно. Можливостями розширення баз даних також вважають створення додаткових масивів даних на диску та шардінг.

Модель ACID підтримує можливість безпечно та надійно виконувати запити, особливо складні запити до великих даних за умов високою завантаженості бази даних [20].

Атомарність – дає гарантію, що всі операції в рамках однієї транзакції будуть виконані, або жодна не виконається. Послідовність гарантує, що база даних буде контролювати початок і кінець всіх транзакцій, що до неї надходять. Ізоляція дозволяє транзакції виконуватися так, начебто вона єдина на всю базу даних, тобто має доступ до всіх ресурсів і вони незмінні у час виконання транзакції.

Довговічність має на увазі неможливість відмінити транзакцію, коли вона була успішно виконана у базі даних. Окрім того, реляційні моделі даних важкі для зберігання та обробки, логування, установки блокування та буферу, або потоків. У більшості випадків, неможливо опрацювати прості операції з даними різних типів, так як вони зберігаються у різних таблицях або базах даних [24].

У розподілених базах даних неможливо зробити об’єднаний запит до кількох баз даних, а при операції оновлення, вставки чи видалення даних – зміни необхідно послідовно заносити до всіх баз даних, які існують у системі. Можуть статися випадки, коли одна з баз даних перестає працювати, що призводить до зростання навантаження у системі та іноді до втрати даних. Усі переваги реляційних моделей полягають у зв’язках між таблицями, але така структура має свої складності, у проєктуванні, розробці та експлуатації. Особливо важко прискорити обробку даних, що зберігаються, при роботі з великими даними, так як час виконання операцій значно зростає.

Через недоліки реляційної моделі виникають інші, нетрадиційні бази даних. Кожна з них повинна була зберігати дані для певного кола задач, так XML бази даних зберігали XML-файли, бази даних у вигляді графів – невеликі обсяги даних. Але найбільш популярними та багатофункціональними стали документні бази даних, сховища типу «ключ-значення» та колоночні бази даних. Їх перевагами були відсутність схеми та структури, вони могли бути

розповсюджені на території та легко масштабувалися за рахунок типу даних, який зберігали [25].

Легке розширення додаванням машин до серверу бази даних дозволяло при проєктуванні системи забувати про проблеми зі зберіганням даних. Кожна колекція зберігала певну частину даних, яка логічно розміщувалася у системі. Швидкість обробки даних можна збільшувати на ходу, додаючи нові машини до кластеру, або збільшуючи потужність єдиного серверу бази даних (горизонтальне або вертикальне масштабування підтримується однаково). Більше того, нереляційні бази даних дозволяють швидко обробляти запити читання та оновлення, які розповсюджені найбільше серед усіх інших типів.

Таким чином, реляційна база даних може створювати гібридні сховища, які зберігають різні типи даних. Мова SQL дозволяє обробляти такі дані та вибирати окремі поля з JSON об’єкта напряму до відповіді на запит. Так як JSON може неповністю зберігати дані всіх полів, а деякі залишати пустими, тому й запити до них будуть повертати NULL. Гібридні моделі також дозволяють використовувати індекси, змінювати JSON структуру на льоту при виконанні запитів, вставляти масиви даних та працювати з його елементами, а також навіть створювати нові документи із структурованих даних. Якщо необхідно зменшити час перетворення об’єкта даних до того, який міг би оброблятися напряму мовою програмування, існує можливість створювати JSON документи у запитах до реляційної бази даних.

Використовуючи команду JSON_OBJECT та комбінуючи її з іншими, можна створювати будь-які об’єкти, навіть використовуючи інші об’єкти. Більше того, такі документи будуть надійними та перевірятися при виконанні SQL операцій [15]. Тобто при операції оновлення даних кожне значення може проходити валідацію, якщо структурувати дані перед записом до бази даних.

Ця функція може бути важлива при збережені важливих конфігураційних даних, або налаштувань, які потім швидко зчитуються для прямого використання у системі. Таким чином, дані різного типу можна зберігати та продуктивно використовувати як в реляційній базі даних, так і NoSQL, але

різниця у підходах полягає у пріоритетах проєктувальника бази даних. Якщо дані повинні бути швидко доступні та узгоджені між собою як всередині бази, так і декількох баз, тоді необхідно використовувати реляційну модуль бази даних. Так як вона може бути гібридною та зберігати JSON документи з можливістю застосування до них SQL запитів, це не відрізняє напівструктуровані дані від звичайних, які зберігаються за структурами таблиць [16].

Реляційні бази даних надійні, а транзакції завжди виконуються, як очікується, завдяки принципам ACID. Їх використання необхідне у великих системах, яким необхідна надійна та узгоджена база даних. Використання тригерів та операцій додає більше можливостей управління даними. Нереляційні бази даних використовуються для швидкості роботи з даними, особливо різнорідними, так як зберігають їх у загальновизнаному форматі та легко передають по мережі. Підтримується більшістю мов програмування без додаткових програмних модулів. Легко масштабується та може не містити значення до деяких полів. Використовуючи принцип BASE системи можуть використовувати базу даних без проєктування, а так, як це вимагається зовнішніми умовами. Використання бази даних може знайти місце в будь-якій системі, як основна чи другорядна модель зберігання даних. Дані можуть зберігатися як на сервері, так і у локальному сховищі на стороні клієнта, у пам’яті або у хмарі [14].

Так, динамічні або конфігураційні дані можуть використовувати сховища типу «ключ-значення» та зберігатися на стороні користувача, а великі дані слід зберігати у хмаровому сховищі або на сервері. Не рекомендується зберігати важливі дані у хмарі, наприклад власні дані користувачів, або банківські акаунти, треба завжди зберігати на власному сервері, що підтверджено на законодавчому рівні [26]. Хмарові сховища краще підходять до розміщення основної бази даних або кластеру розподілених баз даних, так як мають високий рівень доступності, стійкості до відказів та автоматичну масштабованість у випадку, коли це необхідно.

У випадку, коли база даних використовується у якості сервісу (DBaaS), можливо використати додаткові сервіси валідації та збереження даних по категоріям, сортування або фільтрації даних, розподілу потоків даних та контроль за виконанням запитів. При розробці serverless додатка, краще використовувати колонкові бази даних, так як збереження даних не відрізняється від реляційних баз даних, але пошук по базі відбувається, що дозволяє використати менше процесорних хвилин.

Сучасні програмні додатки та інформаційні системи досягли високого рівня розвитку і термін або поняття «архітектура» у застосуванні до них дозволяє грамотно побудувати і сконструювати інформаційну систему в цілому, забезпечуючи її ефективне і надійне функціонування. Архітектура інформаційної системи – концепція, визначальна модель, структуру, виконувані функції і взаємозв'язок компонентів інформаційної системи. У міру розвитку програмних систем все більшого значення набуває їх інтеграція один з одним з метою побудови єдиного інформаційного простору підприємства.

Для того щоб побудувати правильну і надійну архітектуру і грамотно спроєктувати інтеграцію програмних систем необхідно чітко слідувати сучасним стандартам в цих областях. Без цього велика ймовірність, створити архітектуру, яка нездатна розвиватися і задовольняти зростаючим потребам користувачів IT. Класифікація програмних систем за їх архітектурою представляється таким чином [13]:

- централізована архітектура;
- архітектура «файл-сервер»;
- дворівнева архітектура «клієнт-сервер»;
- багаторівнева архітектура «клієнт-сервер»;
- архітектура розподілених систем;
- архітектура Web-додатків;
- сервіс-орієнтована архітектура. Дана інформаційно-довідкова Інтернет-система дошкільного навчально-виховного закладу розроблена як клієнт-серверна система.

Клієнт-серверна система характеризується наявністю двох взаємодіючих самостійних процесів клієнта і сервера, які, в загальному випадку, можуть виконуватися на різних комп'ютерах, обмінюючись даними по мережі. За такою схемою можуть бути побудовані системи обробки даних на основі СУБД, поштові та інші системи.

Додаток на робочій станції виконує важливі функції – відповідає за формування інтерфейсу користувача, логічну обробку даних і за безпосереднє маніпулювання даними. Файловий сервер надає послуги тільки найнижчого рівня – відкриття, закриття і модифікацію файлів. Таким чином, безпосереднім маніпулюванням даними займається кілька незалежних і неузгоджених між собою процесів [6].

1.4. Тенденції розвитку сучасних СУБД

У світі в даний час розроблені і застосовуються донині сотні різних СУБД. Усі СУБД можна класифікувати по виду використовуваної програми , характеру і моделі даних. Найбільш розповсюдженими з них є наступні. Повнофункціональні СУБД, що підтримують локальні і загальні БД з архітектурою «файл-сервер», що мають інтерфейс, що дозволяє створювати, модифікувати структури таблиць, уводити дані, формувати запити, розробляти звіти, виводити їх на печатку: Visual FoxPro, dBASE, Paradox, Access.

Перспективи розвитку архітектур СКБД пов'язані з розвитком концепції обробки нетрадиційних даних та їх інтеграцією, обміном даними з різних СКБД, багатокористувацької технології в локальних мережах. Одна з найважливіших тенденцій розвитку СКБД - розробка «універсальних» СКБД, які здатні інтегрувати в базі традиційні і нетрадиційні дані - тексти, малюнки, звук і відео, сторінки HTML тощо. Це особливо актуально для Web. Сьогодні більшість СКБД вміє працювати не тільки з алфавітно-цифровою інформацією, але і з текстами, аудіо, відео, геоінформацією, XML. Також намітилася тенденція до переміщення неструктурованих даних, раніше збережених поруч з СКБД (у файловій системі), всередину СКБД [7].

Сьогодні нові механізми СКБД (такі як SecureFiles Oracle) дозволять працювати з такими неструктурованими даними не повільніше, а іноді й швидше, ніж при їх зберіганні у файловій системі. Зберігання документів, зображень, відео, аудіо в БД дозволить спростити розробку прикладних програм і підвищить їх якість. Сьогодні бурхливо розвиваються так звані хмарні обчислення (Cloud computing). Ця технологія дуже приваблива для користувачів, тому що можна запросити через Інтернет і одержати у тимчасове користування деякий сервіс для зберігання та обробки даних.

Багатокористувацькі багатофункціональні СУБД, що включають у себе як можливості сервера БД, так і клієнтів: Oracle, Informix, SyBase і ін. Сервери БД призначені для організації центрів обробки даних в архітектурі «клієнт - сервер». Обмін із клієнтськими програмами здійснюється за допомогою операторів SQL. Прикладами є програми: MS SQL Server (MicroSoft), InterBase (Embarcadero), MySQL (Oracle), IBM DB2, Oracle Database, Cache (Inter Systems). Клієнтськими програмами для серверів БД можуть бути повнофункціональні СУБД (Access, FoxPro), електронні таблиці (Excel), текстові процесори (Word), програми електронної пошти. Взаємодія користувача із СУБД відбувається через його чи інтерфейс за допомогою спеціально розробленого додатка, що дозволяє вводити дані, формувати запити до БД і ображати результати пошуку інформації [27].

Додатки можуть бути убудовані в СУБД і незалежні (розроблені за допомогою інших програм). В існуючих СУБД при розробки додатків використовують: - ручне кодування програм (Access, IBM DB2, MS SQL Server Oracle, Cache); - створення текстів додатків за допомогою генераторів (Application Express Oracle). - автоматична генерація готового додатка за допомогою програм візуального програмування (форми Access, Oracle, Cache). Створені будь-яким методом додатки, не можуть виконуватися без СУБД, що аналізує уміст файлів додатка та автоматично створює машинні команди. Цей процес зветься - інтерпретація, використовується в Access, Oracle.

Найбільш розповсюдженими засобами для розробки додатків БД у даний час є: Delphi, C#, C++ Builder, Visual Basic, Java. Основні направлення та тенденції розвитку сучасного програмного забезпечення для проєктування та супроводження БД наступні:

- пошук сучасних моделей зберігання інформації, впровадження нових типів даних в базах;
- розробка нових архітектур СУБД, що забезпечує можливість зберігати і обробляти дані щодо обсягів до петабайт;
- розширення областей застосування БД: опрацювання надвеликих обсягів інформації; розподіленої обробки інформації в мережі;
- забезпеченні інформаційного обслуговування мобільного користувача.

2. ТЕХНІКО–ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ

2.1. Аналіз функціонального інструментарію MySQL

Бази даних організовуються і управляються на основі реляційної моделі мови SQL. Прикладні програми на SQL, як правило, є комбінаціями звичайних програм і операторів SQL. Програми взаємодіють з клієнтами, відображають дані і забезпечують високорівневий напрямок потоку даних. Така модель запропонована з метою збільшення продуктивності баз даних. Додатковою перевагою є незалежність даних методу обробки запиту. Використання SQL дозволяє керувати базою даних при зміні логічних і фізичних схем. Паралельні системи баз даних мають пріоритет над традиційними, так як дозволяють оперувати з великими базами даних в режимі, що підтримує транзакції [29].

MySQL – система управління реляційними базами даних, створена компанією Sun Microsystems (в подальшому і до сьогодення, компанією Oracle). MySQL написаний під десятки видів операційних систем. Це iFreeBSD, OpenBSD, MacOS, OS/2, SunOS, Win9x/00/NTiLinux. Сьогодні MySQL особливо поширена на платформах Linux і Windows. Архітектура MySQL дозволяє зберігати цілі числа довжиною до восьми байтів, строкові значення фіксованої та змінної довжини, числа з плаваючою та фіксованою точкою, а також підтримує всі стандартні запити та модифікатори мови SQL.

Значною перевагою MySQL перед схожими системами управління базами даних є більш висока швидкодія виконання запитів, яка досягається завдяки реалізації функціоналу MySQL мовою C/C++, яка відрізняється більшою швидкістю через свою низькорівневість і роботу напряму із пам’яттю. Проте, не зважаючи на мову реалізації, бази даних, створені в середовищі MySQL добре синхронізуються та взаємодіють із додатками, створеними мовою Java завдяки драйверу JDBC.

MySQL є рішенням для малих і середніх додатків. Входить до складу серверів WAMP, LAMP і в портативні збірки серверів Denver, XAMPP. Зазвичай MySQL використовується як сервер, до якого звертаються локальні або

віддалені клієнти, проте в дистрибутив входить бібліотека внутрішнього сервера, що дозволяє включати MySQL в автономні програми [28].

Принцип роботи СУБД MySQL аналогічний принципу роботи будь СУБД, що використовує SQL (Structured Query Language, мова структурованих запитів) як командної мови для створення/видалення баз даних, таблиць, для поповнення таблиць даними, для здійснення вибірки даних. MySQL, як і будь-яка інша СУБД являє собою програму-сервер, яка знаходиться в пам'яті комп'ютера і обслуговує TCP порт. У випадку з MySQL, номером порту буде число 3306. А клієнтська програма, будь то CGI-додаток на Perl або програмний продукт на C, з'єднується з СУБД з цього порту і посилає йому рядки на SQL. Той у свою чергу їх інтерпретує, виконуючи необхідні дії, і відсилає результати запиту назад клієнтові. Таким способом відбувається спілкування сервера баз даних з клієнтськими програмами. MySQL має розвинену систему доступу до баз даних.

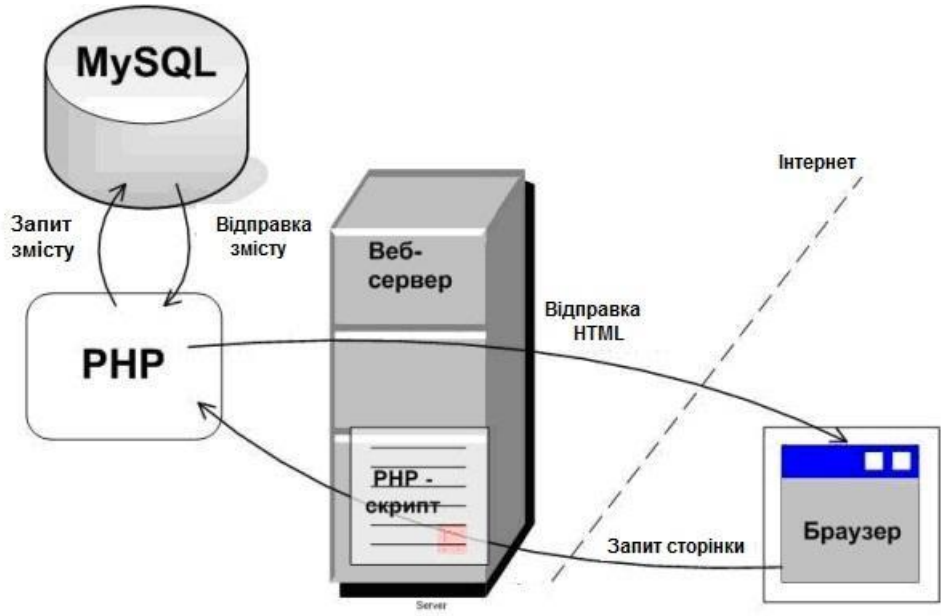


Рисунок 2.1. Схема обробки запитів СУБД MySQL

Користувачеві бази даних може бути наданий доступ до всієї бази даних, окремих таблиць і окремих стовпців таблиць. Є розмежування на дії, які може виробляти користувач із записами. Для організації такої, складною (на перший погляд) структури доступу використовується декілька таблиць в системній базі

даних. На підставі значень цих таблиць налаштовується політика надання доступу [30]. База даних, яку сервер MySQL використовує для зберігання внутрішньою інформації про користувачів, за замовчуванням має ім'я mysql. У цій базі даних певні таблиці для зберігання інформації користувача облікових записів.

Проведемо аналіз функціонального інструментарію MySQL. Для роботи з таблицями звернемося до синтаксису MySQL. CREATE TABLE створює таблицю із заданим іменем. За замовчуванням таблиці створюються в базі даних за замовчуванням за допомогою механізму зберігання InnoDB. Помилка виникає, якщо існує таблиця, якщо немає бази даних за замовчуванням або якщо база даних не існує. MySQL не обмежує кількість таблиць. Базова файлова система може мати обмеження на кількість файлів, що представляють таблиці. Окремі двигуни зберігання можуть встановлювати особливі обмеження для двигуна. InnoDB дозволяє до 4 мільярдів таблиць.

При створенні таблиці CREATE TABLE є можливість вказати наступні характеристики:

Назва таблиці⎪ *Тимчасові таблиці*⎪ *Клонування та копіювання таблиці*⎪ *Типи даних та атрибути стовпців*⎪ *Індекси, foreign-ключі та обмеження перевірки*⎪ *Параметри таблиці*⎪ *Розбиття таблиці*⎪.

При створенні колонок, необхідно вказувати їх типи. Таблиця бази даних містить декілька стовпців із конкретними типами даних, такими як числові або рядкові. MySQL надає більше типів даних, крім простого числового чи рядкового. Кожен тип даних в MySQL може бути визначений за такими характеристиками:

Тип значення, які він представляє.⎪ Простір, який займає, і чи є значення фіксованої або змінної довжини.⎪

Значення типу даних можуть бути індексовані чи ні.⎪ *Як MySQL порівнює значення певного типу даних.*⎪

Для формування більш складних запитів, в мобільному додатку не тільки створені таблиці, але й встановлені зв`язки між ними за допомогою первинних

ключів (primary keys). Первинний ключ таблиці являє собою стовпчик або набір стовпців, які ви використовуєте у своїх найважливіших запитах. Він має асоційований індекс для швидкого виконання запитів.

Ефективність запиту виграє від оптимізації NOT NULL, оскільки вона не може включати будь-які значення NULL. За допомогою системи зберігання InnoDB дані таблиці фізично організовані для здійснення надшвидких пошуків і сортування на основі стовпчика або стовпців первинного ключа. Якщо використовувана таблиця велика і важлива, але в ній немає очевидного стовпчика або набору стовпців, який би використовувався в якості основного ключа, ви можете створити окремий стовпець зі значеннями автоматичного збільшення, який слід використовувати як первинний ключ [33].

Ці унікальні ідентифікатори можуть слугувати вказівниками на відповідні рядки в інших таблицях, коли ви приєднуєте таблиці за допомогою зовнішніх ключів. Деякі таблиці в базі даних MySQL пов'язані між собою. Найчастіше рядок в одній таблиці пов'язаний з декількома рядками в іншій таблиці. Вам потрібен стовпець для з'єднання відповідних рядків у різних таблицях.

У багатьох випадках потрібно включити стовпчик в одну таблицю для зберігання даних, які відповідають даним у стовпчику первинного ключа іншої таблиці. Поширена програма, для якої потрібна база даних з двома пов'язаними таблицями, - це програма замовлення клієнта. Наприклад, одна таблиця містить інформацію про клієнта, таку як ім'я, адреса та номер телефону. Кожен клієнт може мати від нуля до багатьох замовлень. Таким чином, в базі даних серверної частини мобільного додатку визначені допоміжні таблиці, які дозволяють вирішувати проблему при відношеннях N до N.

Так, існує N користувачів додатку, кожен з яких може підписатися на N подій. Завдяки таким зв’язкам можна виконувати більш складні запити і робити вибірки з кількох таблиць.

Так, можна зробити вибірку, що покаже кількість підписників на кожну подію:

SELECT event.id, COUNT(*) AS count FROM `event` INNER JOIN participant on event.id = participant.event_id GROUP BY participant.event_id ORDER BY event.id.

Для прикладу, проаналізуємо доменно-колоночну модель реалізації інформаційної системи швидкісної обробки бази даних MySQL. Для позначення реляційних операцій використали підхід, в якому під $\pi_{*\setminus A}(R)$ розуміється проекція на всі атрибути відношення R, за винятком атрибута A. З допомогою символу «о» позначаємо операцію конкатенації двох кортежів:

$$(x_1, \ldots, x_u) \circ (y_1 \ldots, y_u) = (x_1, \ldots, x_u, y_1 \ldots, y_u).$$

Під R (A, B_1, ..., B_u) розуміємо відношення R з сурогатним ключем A (ідентифікатором цілочислового типу, що однозначно визначає кортеж) і атрибутами B_1, ..., B_u, що представляє собою безліч кортежів довжини u + 1 виду (a, b_1, ..., b_u), де $a \in \mathbb{Z}_{\geq 0}$ та $\forall j \in \{1, \ldots, u\} \, (b_j \in \mathfrak{D}_{B_j})$. Тут $\mathfrak{D}_{B_j}$- домен атрибуту Bj. через r.Bj позначає значення атрибута Bj, через r.A - значення сурогатного ключа кортежу r: r = (rA, r.B_1, ..., r.B_u).

Сурогатний ключ відношення R володіє властивістю $\forall r', r'' \in R \, (r' \neq r'' \Leftrightarrow r'.A \neq r''.A)$. Під адресом кортежу r розуміємо значення сурогатного ключа цього кортежу. Для отримання кортежу відношення R на його адресу використовуємо функцію розіменування $\&_R : \forall r \in R \, (\&_R(r.A) = r)$

.

Далі розглядаємо відношення як множини, а не як мультимножини. Це означає, що якщо при виконанні деякої операції вийшло певне відношення з дублікатами, то до цього відношення за замовчуванням використовується операція видалення дублікатів.

Нехай задано відношення R (A, B, ...), T (R) = n. Нехай на множині $\mathfrak{D}_B$ задано відношення лінійного порядку. Колоночним індексом $I_{R.B}$ атрибута B відношення R будемо називати впорядковане ставлення $I_{R.B}(A, B)$, яке задовольняє наступним властивостям [43]:

$$T(I_{R.B}) = n, \pi_A(I_{R.B}) = \pi_A(R);$$
$$\forall x_1, x_2 \in I_{R.B}(x_1 \le x_2 \Leftrightarrow x_1.B \le x_2.B);$$
$$\forall r \in R(\forall x \in I_{R.B} (r.A = x.A \Rightarrow r.B = x.B)).$$

Умова першого рівняння означає, що множини значень сурогатних ключів (адрес) індексу і індексовані відносини збігаються. Умова другого рівняння означає, що елементи індексу впорядковані в порядку зростання значень атрибута B. Умови третього рівняння означають, що атрибут A елемента індексу містить адресу кортежу відносини R, що має таке ж значення атрибута B, як і у даного елементу колоночного індексу. Із змістовної точки зору колоночний індекс $I_{R.B}$ є таблицею з двох колонок з іменами A і B. Кількість рядків в колоночному індексі збігається з кількістю рядків у індексованій таблиці [42]. Колонка B індексу $I_{R.B}$ включає в себе всі значення колонки B таблиці R (з урахуванням значень, що повторюються), які відсортовані в порядку зростання. Кожен рядок x індексу $I_{R.B}$ містить в колонці A сурогатний ключ (адреса) рядки r в таблиці R, що має таке ж значення в колонці B що і x. На рисунку 2.2 представлені приклади двох різних колоночних індексів для одного і того ж відношення.

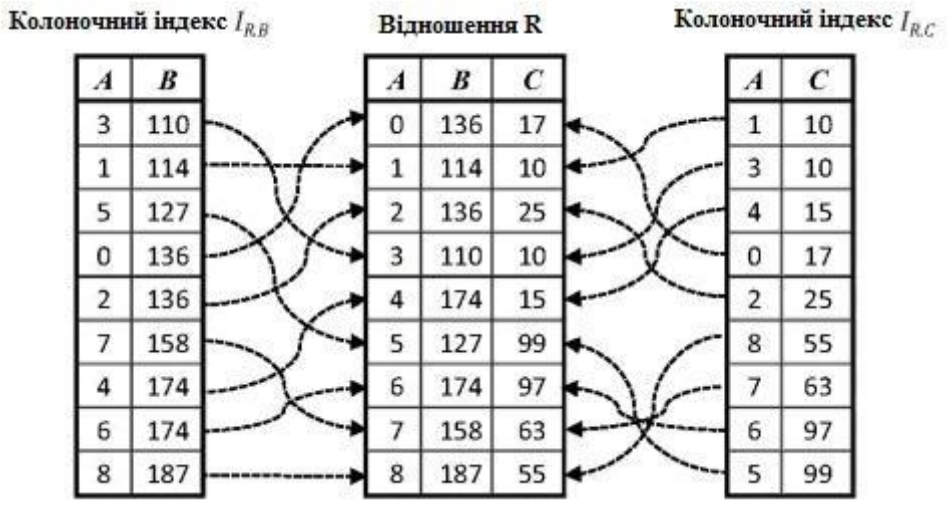


Рисунок 2.2. Колоночний індекс

Нехай на безлічі значень домену $\mathfrak{D}_B$ задано відношення лінійного порядку. Розіб'ємо множину $\mathfrak{D}_B$ на k> 0 неперетинаємих інтервалів:

$$\left.\begin{array}{l} V_0 = [v_0; v_1], V_1 = (v_1; v_2], \ldots, \\ V_{k-1} = (v_{k-1}; v_k]; \\ v_0 < v_1 < \ldots < v_k; \\ \mathfrak{D}_B = \bigcup_{i=0}^{k-1} V_i. \end{array}\right\}$$

Відзначимо, що в випадку $\mathfrak{D}_B = \mathbb{R}$ матимемо $v_0 = -\infty$ та $v_0 = +\infty$. Функція $\varphi_{\mathfrak{D}_B} : \mathfrak{D}_B \to \{0, \ldots, k-1\}$ називається доменною функцією фрагментації для $\mathfrak{D}_B$, якщо вона задовольняє наступній умові:

$$\forall i \in \{0, \ldots, k-1\} \\ (\forall b \in \mathfrak{D}_B (\varphi_{\mathfrak{D}_B}(b) = i \Leftrightarrow b \in V_i))$$

Іншими словами, доменна функція фрагментації відповідає значенням b - номер інтервалу, якому це значення належить.

Нехай заданий колоночний індекс $I_{R.B}$ для відношення R (A, B, ...) з атрибутом B над доменом $\mathfrak{D}_B$ та доменна функція фрагментації $\varphi_{\mathfrak{D}_B}$, функція:

$$\varphi_{I_{R.B}} : I_{R.B} \to \{0, \ldots, k-1\},$$

визначена за правилом:

$$\forall x \in I_{R.B} (\varphi_{I_{R.B}}(x) = \varphi_{\mathfrak{D}_B}(x.B))$$

називається доменно-інтервальною функцією фрагментації для індексу $I_{R.B}$. Іншими словами, функція фрагментації $\varphi_{I_{R.B}} : I_{R.B}$ зіставляє кожному кортежу x з $I_{R.B}$ номер доменного інтервалу, якому належить значенням x.B. Визначимо r перший фрагмент (i = 0, ..., k - 1) індексу $I_{R.B}$ наступним чином:

$$I^i_{R.B} = \{x | x \in I_{R.B};\ \varphi_{I_{R.B}}(x) = i\}$$

Це означає, що в і-тий фрагмент потрапляють кортежі, у яких значення атрибута B належать і-тому доменному інтервалу. Будемо називати таку фрагментацію доменно-інтервального. Кількість фрагментів k будемо називати ступенем фрагментації. Доменно-інтервальна фрагментація має наступні фундаментальними властивостями, що випливають безпосередньо з її визначення [40]:

$$I_{R.B} = \bigcup_{i=0}^{k-1} I_{R.B}^{i};$$

$$\forall i, j \in \{0, \ldots, k-1\} \quad \left(i \neq j \Rightarrow I_{R.B}^{i} \cap I_{R.B}^{j} = \emptyset\right)$$

На рисунку 2.3 схематично зображено фрагментація колоночного індексу, що має ступінь k = 3.

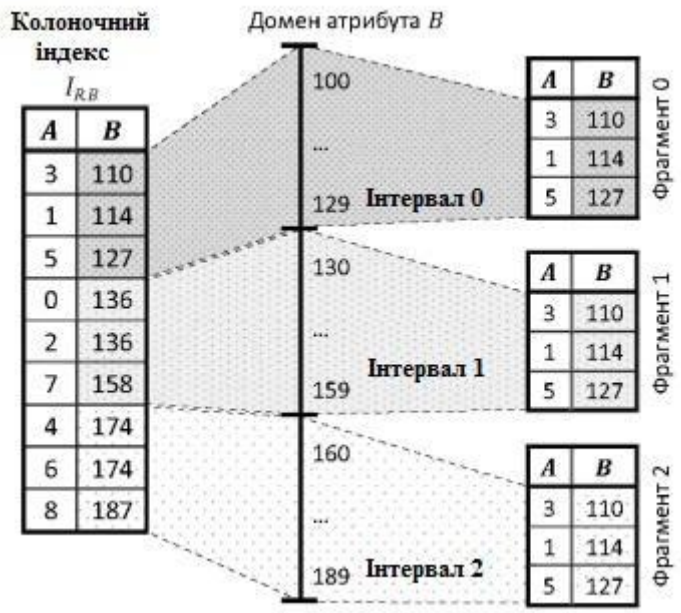


Рисунок 2.3. Фрагмент колоночного індексу

Нехай для відношення R (A, B, C, ...) задані колоночні індекси $I_{R.B}$ та $I_{R.C}$ транзитивних фрагментацією індексу $I_{R.C}$ щодо індексу $I_{R.B}$ називається фрагментацією, що задається функцією $\ddot{\varphi}_{I_{R.C}} : I_{R.C} \to \{0, \ldots, k-1\}$, що задовольняє умові: $\forall x \in I_{R.C}$

$$\ddot{\varphi}_{I_{R.C}}(x) = \varphi_{I_{R.B}}(\sigma_{A=x.A}(I_{R.B}))$$

Транзитивна фрагментація дозволяє розмістити на одному і тому ж вузлі елементи колоночних індексів, що відповідають одному кортежу індексованого відношення.

На базі описаної вище доменно-колоночної моделі представлення даних і методів декомпозиції реляційних операцій розроблена програмна система «колоночний співпроцесор (Ксп)» (Columnar COProcessor сварки) для кластерних обчислювальних систем. В даному розділі описшемо архітектуру і шляхи реалізації колоночного співпроцесора Ксп.

Стовпчик співпроцесор Ксп - це програмна система, призначена для управління розподіленими колоночними індексами, розміщеними в оперативній пам'яті кластерної обчислювальної системи [47]. Призначення Ксп - обчислювати таблиці попередніх обчислень (ТПО) для ресурсномістких реляційних операцій за запитом СУБД. Загальна схема взаємодії СУБД і Ксп зображена на рисунку 2.4.

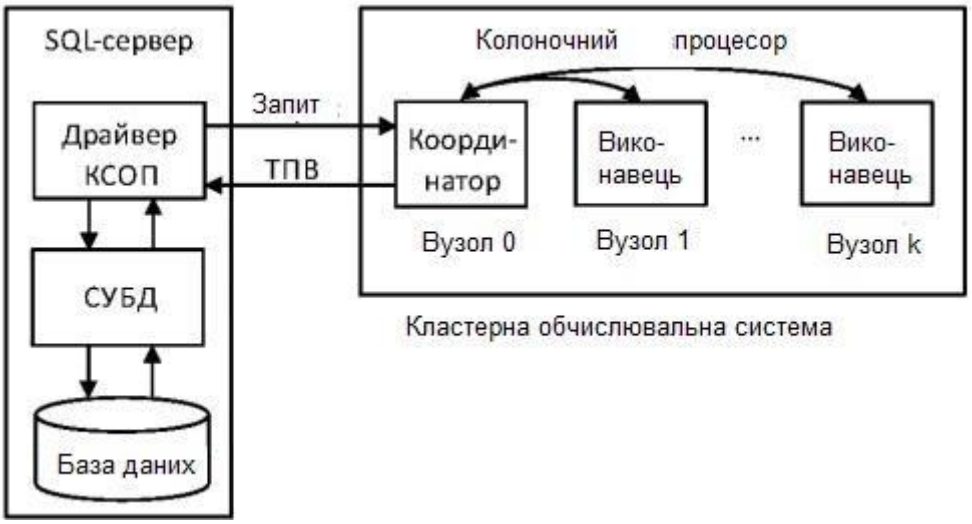


Рисунок 2.4. Схематичне зображення взаємодії SQL-сервера з Ксп

Ксп включає в себе програму «Координатор», що запускається на вузлі обчислювального кластера з номером О, та програму «Виконавець», що запускається на всіх інших вузлах, виділених для роботи Ксп.

На SQL-сервері встановлюється спеціальна програма «Драйвер Ксп», що забезпечує взаємодію з координатором Ксп по протоколу TCP / IP. Ксп працює тільки з даними цілих типів 32 або 64 байта. При створенні колоночного

індексів для атрибутів інших типів, їх значення кодується у вигляді цілого числа або вектора цілих чисел. Ксп підтримує наступні основні операції, доступні СУБД через інтерфейс драйвера Ксп: CreateColumnlndex (створення розподіленого колоночного індексу), Execute (виконання запиту на обчислення ТПВ), Insert (додавання в стовпчик індекс нового кортежу), Transitivelnsert (додавання в стовпчик індекс нового кортежу по транзитивності значенням), Delete (видалення з колоночного індексу кортежу), TransitiveDelete (видалення з колоночного індексу кортежу по транзитивності значенням) [45].

Для організації взаємодії між драйвером і Ксп була розроблена мова CCOPQL (CCOP Query Language), який базується на форматі даних JSON. Кожен стовпчиковий індекс ділиться на фрагменти, які в свою чергу діляться на сегменти. Всі сегменти одного фрагмента розташовуються в стислому вигляді в оперативній пам'яті одного процесорного вузла. Для стиснення сегментів використовувалася бібліотека Zlib [46], що реалізує метод стиснення DEFLATE, що є комбінацією методів Хаффмана і Лемпеля-Зива.

Пояснимо загальну логіку роботи Ксп на простому прикладі. Нехай є база даних з двох відносин R (A, B, D) і S (A, B, C), що зберігаються на SQL сервері (Рисунок 2.5).

Нехай нам необхідно виконати запит:

SELECT D, C

FROM R, S

WHERE R.B = S.B AND C <13.

Припустимо, що Ксп має тільки два вузли-виконавці і на кожному вузлі є три процесорних ядра (процесорні ядра на рисунку 3.5 промарковані позначками P_{11}, ..., P_{23}) - Покладемо, що атрибути R.B і S.B визначені на домені цілих чисел з інтервалу [0; 120). Сегментні інтервали для R.B і S.B визначимо наступним чином: [0; 20), [20; 40), [40; 60), [60; 80), [80; 100), [100; 120).

В якості фрагментних інтервалів для R.B і S.B зафіксуємо: [0; 59] і [60,119]. Нехай атрибут S. визначено на домені цілих чисел з інтервалу [0; 25].

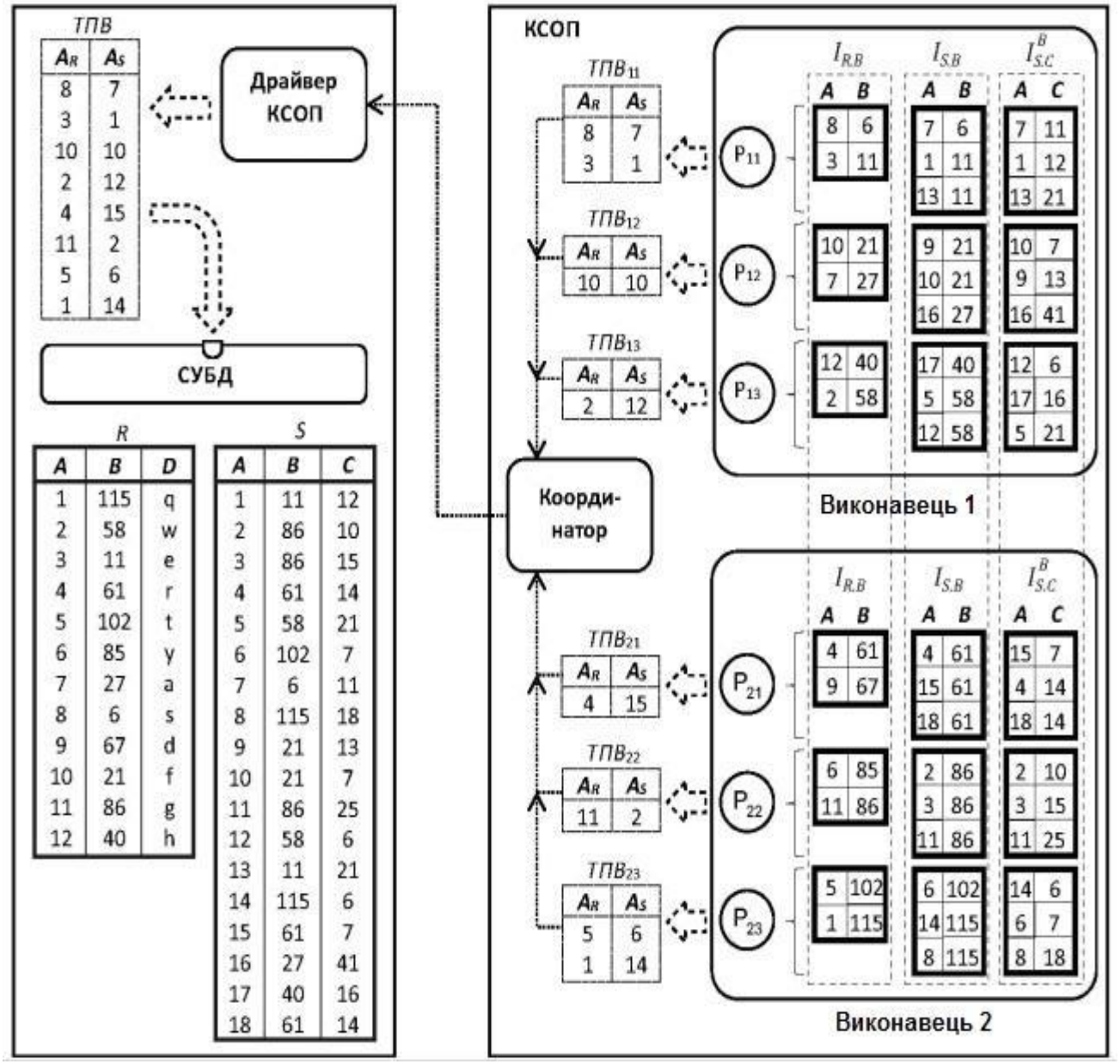


Рисунок 2.5. Схематичне зображення обчислення ТПО з використанням Ксп

Спочатку адміністратор бази даних за допомогою драйвера Ксп створює для атрибутів R.B і S.B розподілені колоночні індекси $I_{R.B}$ і $I_{S.B.}$ Потім для атрибута S. створюється розподілений стовпчик індекс $I^{B}_{S.C}$, який фрагментується і сегментується транзитивно щодо індексу $I_{S.B}$ Розподілені колоночного індекси $I_{R.B}$, $I_{S.B}$ та $I^{B}_{S.C}$ зберігаються в оперативній пам'яті вузлів-виконавців. Таким чином, ми отримуємо розподіл даних всередині Ксп, наведене на рисунку 2.5. При надходженні SQL запиту він перетвориться

драйвером Ксп в план, який визначається наступним виразом реляційної алгебри:

$$\pi_{I_{R.B}.A \to A_R, I_{S.B}.A \to A_S}(I_{R.B} \underset{I_{R.B}.B = I_{S.B}.B}{\bowtie} (I_{S.B} \bowtie \sigma_{C<13}(I^B_{S.C})))$$

При виконанні драйвером операції Execute вказаний запит передається координатору Ксп у вигляді оператора CCOPQL в форматі JSON. Запит виконується незалежно процесорними ядрами вузлів-виконавців над відповідними групами сегментів. При цьому за рахунок доменної фрагментації і сегментації не потрібні обміни даними як між вузлами-виконавцями, так і між процесорними ядрами одного вузла.

Кожне процесорне ядро обчислює свою частину ТПВ, яка пересилається на вузол-координатор. Координатор об'єднує фрагменти ТПВ в єдину таблицю і пересилає її драйверу, який виконує матеріалізацію цієї таблиці у вигляді порівнянь в базі даних, що зберігається на SQL сервері. Після цього SQL сервер замість вихідного SQL оператора, виконує наступний оператор [51]:

SELECT D, C

FROM

R INNER JOIN (

ТПО INNER JOIN S ON (S.A = ТПО.AS)

) ON (R.A = ТПО.AR).

При цьому використовуються звичайні кластеризовані індекси у вигляді B-дерев, що заздалегідь побудовані для атрибутів R.A і S.A.

Колоночний співпроцесор Ксп був реалізований з використанням апаратно-незалежних технологій MPI та OpenMP.

Опис структур даних. Декомпозиція полягає в розбитті алгоритму виконання операції на окремі підзадачі для подальшого обміну даними.

Декомпозиція природного з'єднання. Розглянемо декомпозицію операції природного з'єднання вигляду $\pi_{*\setminus A}(R) \bowtie \pi_{*\setminus A}(S)$. Нехай задані два відношення R (A, B_1, ..., B_u, C_1, ..., C_v) та S (A, B_1, ..., B_u, D_1, ..., D_w). Нехай є два набори

колоночних індексів але атрибутам B_1, ..., B_u $I_{R.B1}$, ..., $I_{R.BU}$ $I_{S.B1}$, ..., $I_{S.Bu}$. Нехай для всіх цих індексів задана доменно-інтервальна фрагментація ступеня k:

$$I_{R.B_j} = \bigcup_{i=0}^{k-1} I^i_{R.B_j};\ I_{S.B_j} = \bigcup_{i=0}^{k-1} I^i_{S.B_j}.$$

Припустимо:

$$P^i_j = \pi_{I^i_{R.B_j}.A \to A_R,\ I^i_{S.B_j}.A \to A_S}(I^i_{R.B_j} \underset{I^i_{R.B_j}.B_j = I^i_{S.B_j}.B_j}{\bowtie} I^i_{S.B_j})$$

для всіх i = 0, ..., k - 1. Визначимо:

$$P = \bigcup_{i=0}^{k-1} P^i.$$

побудуємо відношення:

$$Q\ (B_1, \ldots, B_u, C_1, \ldots, C_v, D_1, \ldots, D_w)$$

наступним чином:

$$Q = \{(\&_R(p.A_R).B_1, \ldots, \&_R(p.A_R).B_u, \&_B(p.A_B).C_1, \ldots, \&_B(p.A_B).C_v, \&_S(p.A_S).D_1, \ldots, \&_S(p.A_S).D_w) | p \in P\}.$$

Тоді $Q = \pi_{*\setminus A}(R) \bowtie \pi_{*\setminus A}(S)$. Приклад розрахунку операції природного сполучення двох відношень з використанням розподілених колоночних індексів перевірено також рядом інших науковців і отримані результати засвідчують їх правдивість.

Декомпозиція операції угруповання. Розглянемо декомпозицію операції угрупування виду $\gamma_{B,C_1,\ldots,C_u,\mathrm{agrf}(D_1,\ldots,D_w)\to F}(R)$. Нехай задано відношення

$$R\,(A, B, C_1, \ldots, C_u, D_1, \ldots, D_w, \ldots)$$

з сурогатним ключем A. Нехай для атрибутів D_1, ..., D_w задана агрегуюча функція agrf. Нехай є колоночний індекс $I_{R.B}$. Нехай також є колоночні індекси:

$I_{R.C1}$, ..., $I_{R.Cu}$; $I_{R.D1}$, ..., $I_{R.Dw}$. Нехай для індексу $I_{R.B}$ задана доменно-інтервальна фрагментація ступеня k:

$$I_{R.B} = \bigcup_{i=0}^{k-1} I^i_{R.B}$$

Нехай для індексів $I_{R.C1}$, ..., $I_{R.Cu}$ та $I_{R.D1}$, ..., $I_{R.Dw}$ задана транзитивній відносно $I_{R.B}$ фрагментація:

$$\forall j \in \{1,\dots,u\}\left(I_{R.C_j} = \bigcup_{i=0}^{k-1} I^i_{R.C_j}\right) \quad \forall j \in \{1,\dots,w\}\left(I_{R.D_j} = \bigcup_{i=0}^{k-1} I^i_{R.D_j}\right)$$

Припустимо:

$$P_i = \pi_{A,\,F}\Big(\gamma_{\min(A)\to A,B,C_1,\dots,C_u,\mathrm{agrf}(D_1,\dots,D_w)\to F}\Big(I^i_{R.B} \bowtie I^i_{R.C_1} \bowtie \dots \bowtie I^i_{R.C_u} \bowtie I^i_{R.D_1} \bowtie \dots \bowtie I^i_{R.D_w}\Big)\Big)$$

для всіх i = 0, ..., k - 1. Визначимо $P = \bigcup_{i=0}^{k-1} P_i$. Побудуємо відношення Q (B, C_1, ..., C_u, F) наступним чином:

$$Q = \{(\&_R(_P A).B, \&_R(_P A).C_1, \dots, \&_R(_P A).C_U, {}_P F)\ I_P \in P\}.$$

Тоді $Q = \gamma_{B,C_1,\dots,C_u,\mathrm{agrf}(D_1,\dots,D_w)\to F}(R)$. Доведення коректності описаної декомпозиції операції угруповання наводиться також в роботі інших науковців.

Декомпозиція операції перетину. Розглянемо декомпозицію операції перетину виду $\pi_{B_1,\dots,B_u}(R) \cap \pi_{B_1,\dots,B_u}(S)$. Нехай задані дві відносини R (A, B_1, ..., B_u) та S (A, B_1, ..., B_u), що мають однаковий набір атрибутів. Нехай є два набори колоночних індексів по атрибутам B_1, ..., B_u: $I_{R.B1}$, ..., $I_{R.Bp}$, $I_{S.B1}$, ..., $I_{s.Bu}$. Нехай для всіх цих індексів задана доменно-інтервальною фрагментацією ступеня k [52]:

$$I_{R.B_j} = \bigcup_{i=0}^{k-1} I^i_{R.B_j}; I_{S.B_j} = \bigcup_{i=0}^{k-1} I^i_{S.B_j}.$$

Припустимо:

$$P_j^i = \pi_{I^i_{R.B_j}.A \to A_R,\ I^i_{S.B_j}.A \to A_S}(I^i_{R.B_j} \underset{(I^i_{R.B_j}.B_j = I^i_{S.B_j}.B_j)}{\bowtie} I^i_{S.B_j})$$

для всіх i = 0, ..., k - 1 і j = 1, ..., u. Визначимо $P_j = \bigcup_{i=0}^{k-1} P_j^i$. Припустимо $P = \bigcap_{j=1}^{u} P_j$. Побудуємо відношення Q (A, B_1, ..., B_u) наступним чином:

$$Q = \{r | r \in R \wedge r.A \in \pi_{A_R}(P)\}$$

Тоді $\pi_{B_1,\ldots,B_u}(Q) = \pi_{B_1,\ldots,B_u}(R) \cap \pi_{B_1,\ldots,B_u}(S)$. Доведення коректності описаної декомпозиції операції перетину наведено в наукових дослідженнях інших науковців.

З використанням описаної методики ми можемо виконувати декомпозицію операцій проекції, вибору, вилучення дублікатів та об'єднання баз даних.

Таким чином, MySQL використовують у таких ситуаціях:

- при розподілених операціях, коли функціоналу SQLite (інша популярна система) не вистачає;
- для роботи з інтернет-сторінками та веб-додатками, оскільки MySQL є найбільш зручною СУБД для цієї сфери застосування;
- під час роботи зі специфічним проєктом, де функціонал MySQL дає оптимальний результат.

2.2. Особливості використання Google BigQuery

Технологія BigQuery кардинально змінила спосіб представлення корпоративних даних. Будучи спочатку призначеною для роботи з гігантськими наборами даних, BigQuery стала однією з найкращих платформ для аналізу та вивчення даних. BigQuery – хмарне, безсерверне сховище корпоративних даних,

яке допомагає розробляти, впроваджувати та масштабувати керування даними з інтелектуальних додатків для цифрової трансформації [50].

Механізм запитів в Google BigQuery здатний виконувати запити SQL на терабайтах даних за лічені секунди, а на петабайтах – за лічені хвилини. Щоб отримати таку продуктивність, не доведеться організовувати та підтримувати будь-яку інфраструктуру та створювати або перебудовувати індекси.

```
SELECT
 EXTRACT(YEAR FROM starttime) AS year,
EXTRACT(MONTH FROM starttime) AS month,
COUNT(starttime) AS number_one_way
FROM
`bigquery-public-data.new_york_citibike.citibike_trips`
WHERE
 start_station_name != end_station_name
GROUP BY year, month
ORDER BY year ASC, month ASC
```

BigQuery підтримує свій високоефективний формат колонкового зберігання, що робить методологію ETL особливо привабливою. Конвеєр обробки даних, зазвичай реалізований на основі Apache Beam або Apache Spark, витягує необхідні вихідні дані (потокових даних або пакетних файлів), перетворює вилучені дані, готуючи їх до очищення або агрегування, а потім завантажує їх у BigQuery.

Хоча створення конвеєра ETL в Apache Beam або Apache Spark є дуже поширеним, його можна створити виключно в BigQuery. Оскільки BigQuery відокремлює обчислення від сховища, SQL-запити BigQuery можна виконувати для файлів у форматі CSV (а також JSON та Avro), які зберігаються у вихідному вигляді у хмарному сховищі Google Cloud Storage; ця можливість називається федеративним запитом. За допомогою федеративних запитів можна отримати дані, виконуючи запити SQL до Google Cloud Storage, перетворювати дані всередині цих запитів SQL і зберігати результати у власних таблицях BigQuery.

Якщо перетворення не потрібне, BigQuery може безпосередньо завантажувати стандартні формати, такі як CSV, JSON або Avro, у своє сховище – робочий процес EL (витяг та завантаження). Такий підхід, коли дані завантажуються безпосередньо у сховище, використовується тому, що зберігання даних у власному сховищі забезпечує найбільшу ефективність запитів [48].

BigQuery може приймати як пакетні, так і потокові дані. Дані можна передавати до BigQuery безпосередньо через REST API. Часто користувачі, яким потрібно перетворити дані, наприклад, проводячи обчислення з тимчасовим вікном, використовують конвеєри Apache Beam, які виконує сервіс Cloud Dataflow. І навіть при передачі потокових даних до BigQuery можна запросити їх. Наявність загальної інфраструктури запитів для архівних (пакетних) та поточних (потокових) даних відкриває широкі можливості та спрощує багато процесів.

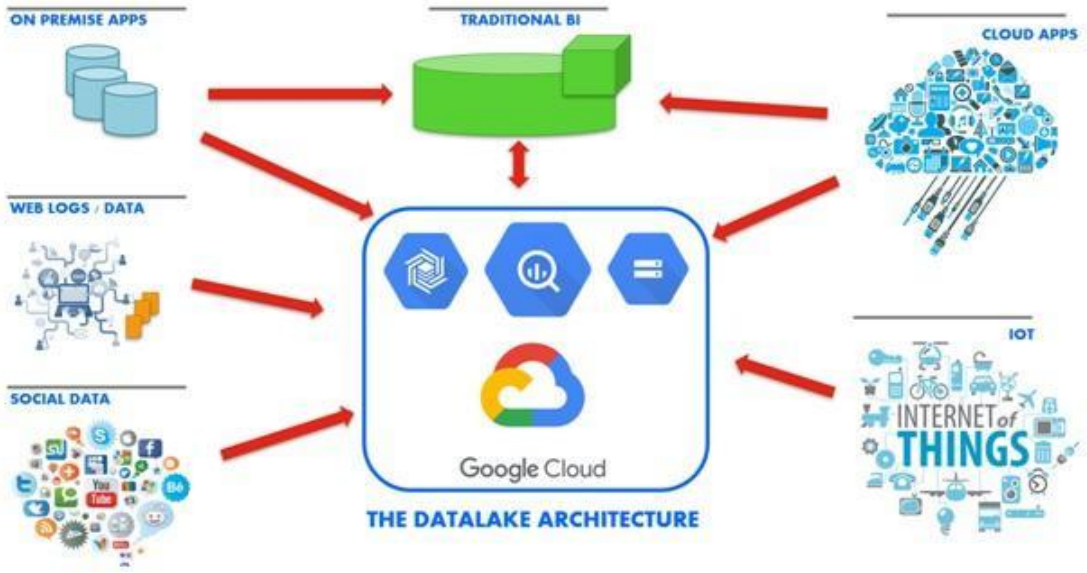


Рисунок 2.6. Схематичне зображення головних областей застосування Google Cloud

Дані BigQuery автоматично шифруються як при зберіганні, так і при передачі. BigQuery піклується про безпеку розрахованих на багато користувачів запитів і ізоляції завдань. Є змога організувати спільний доступ до власних наборів даних за допомогою сервісу Google Cloud Identity and Access Management (IAM), а також застосовувати до наборів даних (а також таблиць та

подань у них) різні заходи безпеки, залежно від того, чи потрібна вам відкритість , можливість аудиту чи конфіденційність [49].

При побудові запитів в BigQuery найчастіше використовуються такі групи функцій: функції агрегування даних (Aggregate function), функції для роботи з датами (Date function), функції для роботи із рядками (String function) та функції для роботи з підмножиною даних або віконні функції (Window function).

За допомогою віконних функцій можна агрегувати дані в розрізі груп, не використовуючи оператор JOIN для об'єднання кількох запитів. Наприклад, розрахувати середній дохід у розрізі рекламних кампаній, кількість транзакцій у розрізі пристроїв. Додавши ще одне поле у звіті можна легко дізнатися, наприклад, частку доходу від рекламної кампанії на Black Friday або транзакцій, зроблених з мобільного додатка. Разом з кожною функцією у запиті необхідно прописувати вираз OVER, який визначає межі вікна. OVER містить 3 компоненти, з якими ви можете працювати:

- PARTITION BY - визначає ознаку, за якою ви ділитимете вихідні дані на підмножини, наприклад PARTITION BY clientId, DayTime.
- ORDER BY - визначає порядок рядків у підмножині, наприклад, ORDER BY hour DESC.
- WINDOW FRAME - дозволяє обробляти рядки всередині підмножини за певною ознакою. Наприклад, можна порахувати суму не всіх рядків у вікні, а лише перших п'яти перед поточним рядком.

Система зберігання у BigQuery заснована на ідеї використання абстракції таблиці замість абстракції файлу під час роботи зі структурованим сховищем. Деякі хмарні системи обробки даних з відкритим вихідним кодом пропонують користувачам формат файлу, що дозволяє керувати розмірами файлів та забезпечувати узгодженість схеми.

Сховища дозволяють створювати файли відповідного розміру для зберігання статичних даних, але, як відомо, підтримувати оптимальний розмір файлів для даних, які з часом змінюються дуже складно. Так само важко підтримувати узгодженість схеми за наявності великої кількості файлів зі

схемами, що передбачають самоопис (наприклад, Avro або Parquet), зазвичай кожне оновлення програмного забезпечення в системах, що виробляє ці файли, призводить до змін схеми [44].

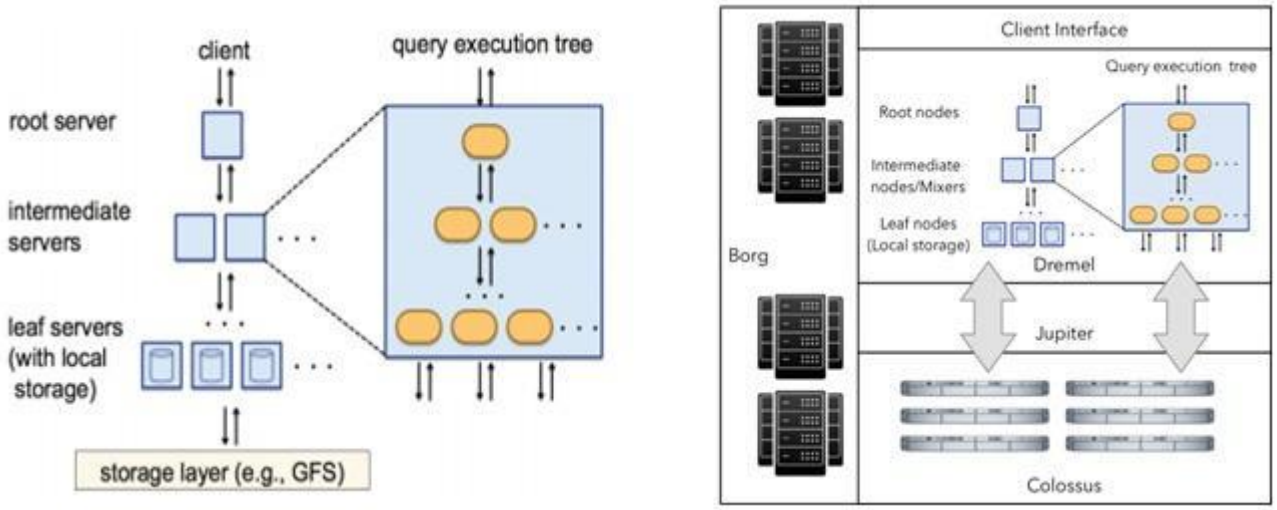


Рисунок 2.7. Архітектура BigQuery

Ще одна перевага BigQuery в управлінні власним сховищем: можливість збільшення швидкості роботи без зусиль кінцевого користувача. Наприклад, покращення у форматах зберігання можуть автоматично застосовуватися до даних користувача. Аналогічно, покращення в інфраструктурі зберігання негайно відбиваються на роботі системи. Оскільки сховищем керує лише BigQuery, користувачам не потрібно турбуватися про резервне копіювання або реплікацію [41]. Все, від оновлень та реплікації до резервного копіювання та відновлення, виконується системою керування сховищем автоматично. Однією з ключових переваг роботи зі структурованим сховищем на рівні абстракції таблиці (а не файлу) та простого управління зберіганням цих таблиць є можливість для BigQuery підтримувати відповідні функції, наприклад, DML.

У BigQuery підтримуються чотири основні CRUD-операції:

– Create (Створення) додає нові записи. Здійснюється за допомогою операцій завантаження, SQL-оператора INSERT та інтерфейсу потокової вставки. За допомогою SQL-операторів, які є частиною мови визначення даних (Data Definition Language, DDL), що підтримується в BigQuery,

також можна створювати об'єкти баз даних, такі як таблиці, уявлення та моделі машинного навчання. Нижче ми розглянемо приклади кожної з наведених можливостей.

- Read (читання) вилучає записи. Здійснюється за допомогою SQL-оператора SELECT та масового зчитування API.
- Update (Зміна) змінює наявні записи. Здійснюється за допомогою SQL-операторів UPDATE і MERGE, які є частиною мови маніпулювання даними, що підтримується в BigQuery (Data Manipulation Language, DML). Платформа BigQuery є аналітичним інструментом і не призначена для частих оновлень.
- Delete (Видалення) видаляє наявні записи. Здійснюється за допомогою SQL-оператора DELETE, який також є частиною мови DML, що підтримується в BigQuery.
- Масив структур STRUCT - це група полів, які у певному порядку. Полям можна давати імена (якщо цього не зробити, BigQuery надасть їм свої імена), і ми радимо використовувати їх для покращення читаності запитів:

```
SELECT
 [
STRUCT('male' as gender, [9306602, 3955871] as numtrips)
, STRUCT('fe ' as gender, [3236735, 1260893] as numtrips)
] AS bikerides
```

Робота з масивами. Маючи масив, можна визначити його довжину та витягти з нього окремі елементи:

```
SELECT
 ARRAY_LENGTH(bikerides) as num_items
, bikerides[ OFFSET(0) ].gender as first_gender
FROM
(SELECT
[
 STRUCT('male' as gender, [9306602, 3955871] as numtrips)
```

, STRUCT('female' as gender, [3236735, 1260893] as numtrips)

] AS bikerides)

Розгортання масиву. У запиті

SELECT

[

STRUCT('male' as gender, [9306602, 3955871] as numtrips)

, STRUCT('female' as gender, [3236735, 1260893] as numtrips)

]

оператор SELECT поверне єдиний рядок, що містить масив, тому обидва підлоги є частиною одного рядка (зверніть увагу на стовпець «Row»):

- UNNEST - це функція, яка повертає елементи масиву у вигляді рядків, тому UNNEST можна застосувати до масиву результатів, щоб отримати рядок, який відповідає кожному елементу масиву:

  *SELECT * від UNNEST(*

  [

  STRUCT('male' as gender, [9306602, 3955871] as numtrips)

  , STRUCT('female' as gender, [3236735, 1260893] as numtrips)

  ])

- UNNEST є проміжним джерелом (from_item) - до нього можна застосувати оператор SELECT. Також можна вибрати лише частини масиву. Наприклад, можна отримати лише стовпець numtrips:

SELECT numtrips from UNNEST(

[

STRUCT('male' as gender, [9306602, 3955871] as numtrips)

, STRUCT('female' as gender, [3236735, 1260893] as numtrips)

])

- CROSS JOIN - це перехресне з'єднання, або декартове твір, що не має умови з'єднання. Об'єднуються всі записи з обох проміжних джерел. Таке з'єднання ми отримали б, якби умова з'єднання INNER JOIN завжди оцінювалося як справжнє:

```
WITH winners AS (
 SELECT 'John' as person, '100m' as event
 UNION ALL SELECT 'Hiroshi', '200m'
 UNION ALL SELECT 'Sita', '400m'
),
gifts AS (
 SELECT 'Google Home' as gift, '100m' as event
 UNION ALL SELECT 'Google Hub', '200m'
 UNION ALL SELECT 'Pixel3', '400m'
)
SELECT winners.*, gifts.gift
FROM winners
JOIN gifts
```

Веб-інтерфейс BigQuery дозволяє зберігати та обмінюватися запитами. Це зручно: ви можете надіслати колегам посилання на текст запиту, а ті негайно виконати його. Але потрібно враховувати, що якщо у когось є запит, це не означає, що він гарантовано зможе виконати його, бо він має може бути недостатньо прав для доступу до даних [39].

BigQuery підтримує кілька типів даних для зберігання чисел, рядків, значення часу, географічних координат, структурованих та напівструктурованих даних:

- INT64. Це єдиний цілий тип. Для представлення речових чисел використовується тип FLOAT64, а логічних значень – тип BOOL.
- NUMERIC. Тип NUMERIC забезпечує точне представлення дійсних чисел, що містять до 38 цифр та 9 десяткових знаків після коми, та підходить для точних розрахунків, наприклад, у сфері фінансів.
- STRING. Цей тип даних є послідовністю символів Юнікод змінної довжини. А для представлення однобайтових послідовностей символів (не Юнікод) змінної довжини використовується тип BYTES.
- TIMESTAMP. Представляє абсолютне значення часу.

– DATETIME. Представляє календарну дату та час. Також доступні окремі типи DATE та TIME.
– GEOGRAPHY. Тип GEOGRAPHY представляє точки, лінії та полігони на поверхні землі.

BigQuery може завантажувати дані з файлів CSV, однак це не найефективніший і виразніший спосіб (наприклад, з файлів CSV не можна завантажувати масиви та структури), при цьому експортують дані до іншого формату. Найбільш ефективним та виразним є формат Avro (https://avro.apache.org/). Це двійковий формат із самоописом. Дані в цьому форматі розбиваються на блоки і всі блоки стискаються. Завдяки цьому можна паралельно завантажувати дані у форматі Avro та експортувати до цього формату [38].

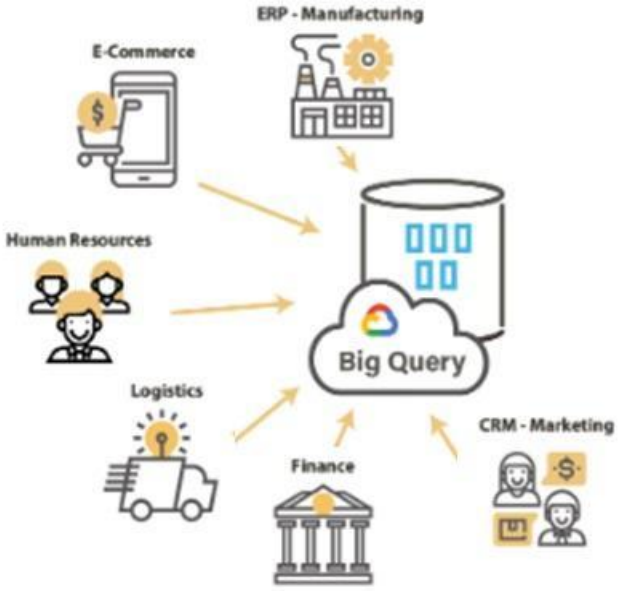


Рисунок 2.8. Основні сфери застосування системи BigQuery

Крім того, оскільки блоки стискаються, файли виходять менше, ніж обсяг даних у них. Формат Avro є ієрархічним, може представляти вкладені та повторювані поля та підтримується платформою BigQuery. Організувати збереження подібних даних у файлах CSV набагато складніше. Оскільки файли Avro включають самоопис, їх не потрібно вказувати у схемі. Єдиним недоліком формату Avro є його непридатність для читання людиною.

Якщо для користувача важливими є зручність для читання та виразність, то використовують для зберігання даних рядковий формат JSON. Формат JSON підтримує можливість зберігання ієрархічних даних, але вимагає представлення двійкових даних у кодуванні base-64. Однак формат JSON пропонує ширші можливості, ніж CSV, тому що ім'я кожного поля повторюється у кожному рядку [37].

Формат Parquet став підтримуватись у BigQuery відносно недавно. Він заснований на оригінальному форматі Google Dremel ColumnIO, і так само, як Avro, є бінарним, блочно-орієнтованим і компактним і здатний представляти ієрархічні дані.

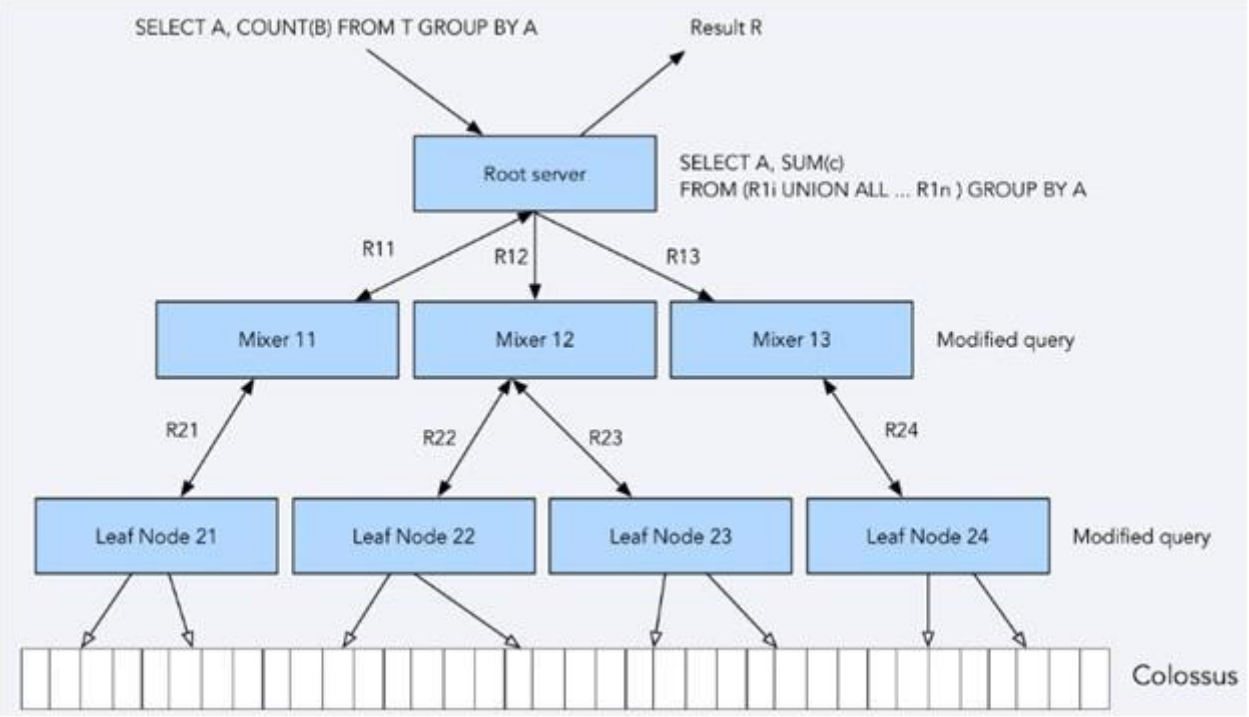


Рисунок 2.9. Приклад дерева сервірування Dremel

Однак, на відміну від формату Avro, в якому дані зберігаються строково, запис за записом формат Parquet зберігає дані по стовпцях. Файли колонки краще підходять для читання підмножини стовпців, проте в процесі завантаження даних потрібно читати всі стовпці, тому колонкові формати дещо менш ефективні при завантаженні даних. Проте колонковий формат робить Parquet кращим, ніж Avro, для федеративних запитів, про які ми поговоримо

пізніше. Ще один відкритий колонковий формат – Optimized Row Columnar (ORC). Формат ORC порівняний з Parquet за продуктивністю та ефективності.

При використанні форматів без внутрішнього стиснення, таких як CSV та JSON, слід подумати про стиснення файлів за допомогою gzip. Стислі файли передаються швидше і займають менше місця, але вони повільніше завантажуються у BigQuery [36]. Чим повільніше мережа, тим вигідніше стиснення. Якщо користувач працюєте в повільній мережі, або у користувача багато файлів, або вони дуже великі, то багатопоточне завантаження даних можна налаштувати за допомогою gsutil cp. Після збереження всіх даних у хмарному сховищі Google Cloud Storage ви зможете завантажити їх у BigQuery звідти за допомогою команди bq load:

*gsutil -m cp *.csv gs://BUCKET/some/location*

bqload ... gs://BUCKET/some /location/.csv*

У цьому експерименті представлені різні плюси та мінуси, пов'язані зі стисненням та розміщенням даних з оціночними параметрами кортежів у хмарному сховищі Cloud Storage перед викликом команди bq load. Отримані результати, звичайно, відрізнятимуться і залежатимуть від характеристик мережі та самих завантажених даних. Тому проведіть аналогічні дослідження у себе та виберіть метод, що забезпечує найкращу ефективність. У разі розміщення файлу в хмарному сховищі Google Cloud Storage вам доведеться заплатити за зберігання даних принаймні до завершення завдання завантаження в BigQuery. Проте витрати на зберігання, як правило, несуттєві, тому для цього набору даних та мережевого підключення найкращим варіантом є розміщення стислих даних у хмарному сховищі з подальшим завантаженням їх звідти. Незважаючи на те, що несжаті файли завантажуються в BigQuery швидше, час передачі файлів по мережі зменшує переваги швидшого завантаження.

Ресурс підключення до BigQuery у Google Sheets дозволяє запитувати таблиці BigQuery та заповнювати електронні таблиці результатами запитів. Це може стати в нагоді для обміну даними з нетехнічними користувачами. Більшість організацій майже всі офісні працівники знають, як

читати/інтерпретувати електронні таблиці. Їм не потрібно бути в курсі всіх тонкощів BigQuery або SQL, щоб використовувати Google Sheets і працювати з даними на аркуші.

BigQuery та Google Sheets здатні зберігати та надавати доступ до табличних даних. Проте, BigQuery - це насамперед сховище аналітичних даних, а Google Sheets насамперед є інтерактивним документом. Використання Sheets, для дослідження та побудови діаграм робить завантаження даних BigQuery у Sheets дуже функціональною можливістю. Проте існує технічне обмеження на розмір наборів даних BigQuery, які можна завантажити у Sheets.

Наприклад, у BigQuery зберігається інформація про питання, відповіді та користувачів сайту Stack Overflow. Навіть якщо використовувати BigSheets, ці петабайтні набори даних дуже великі, щоб їх можна було завантажити безпосередньо в Google Sheets. Однак є можливість писати запити, що з'єднують невеликі набори даних у Sheets з гігантськими наборами даних у BigQuery та опрацьовують результат.

Служба передачі даних BigQuery Data Transfer Service дозволяє запланувати періодичне завантаження даних із різних джерел у BigQuery. Як і багато інших можливостей BigQuery, BigQuery Data Transfer Service доступна з веб-інтерфейсу, інструменту командного рядка або через REST API. Щоб ви могли спробувати на прикладі, ми покажемо спосіб з використанням командного рядка.

Після налаштування передачі даних BigQuery автоматично завантажуватиме дані за вказаним вами розкладом. Але якщо з вихідними даними виникне проблема, ви зможете ініціювати зворотну передачу даних для відновлення після будь-яких збоїв або аварій. Така передача називається оновленням, і вона може запуститися з веб-інтерфейсу. Служба Data Transfer Service підтримує завантаження даних з багатьох програм типу програмне забезпечення як послуга (Software as a Service, SaaS), таких як Google Ads, Google Play, Amazon Redshift та YouTube, а також з Google Cloud Storage. Далі ми подивимося, як налаштувати звичайне завантаження файлів з хмарного

сховища Cloud Storage, і водночас відзначимо відмінності від передачі даних із набору даних SaaS на прикладі звітів про канали на YouTube.

Для програмного доступу до BigQuery рекомендується використовувати клієнтську бібліотеку Google Cloud Client Library на вибраній мові програмування. Клієнтська бібліотека була доступна для семи мов програмування: Go, Java, Node.js, Python, Ruby, PHP та C++. Знання REST API допоможе зрозуміти, що відбувається всередині після відправлення запиту до служби BigQuery, але клієнтська бібліотека BigQuery більш практична.

JSON/REST - це архітектурний стиль проєктування розподілених служб, запити до яких не мають стану (тобто сервер не підтримує стану сеансу або контекст; натомість вся необхідна інформація передається в самому запиті), причому запити та відповіді мають текстовий формат JSON. Оскільки протокол HTTP не підтримує стану, служби REST особливо добре підходять для надання послуг через Інтернет. Повідомлення у форматі JSON відображаються безпосередньо в об'єктах пам'яті на таких мовах, як JavaScript та Python.

Програмні інтерфейси REST API створюють ілюзію того, що об'єкти, на які вони посилаються, є статичними файлами і для них підтримуються операції Create (створити), Read (прочитати), Update (змінити), Delete (видалити) - CRUD, що прямо відповідають дієсловам HTTP. Наприклад, щоб створити таблицю в BigQuery, потрібно надіслати HTTP-запит POST, щоб дослідити таблицю - HTTP-запит GET, щоб змінити її - HTTP-запит PATCH, а щоб видалити - HTTP-запит DELETE. Є також такі методи, як Query, які не мають точного аналога в наборі операцій CRUD, тому їх часто називають методами виклику віддалених процедур (Remote Procedure Call, RPC).

Іноді недостатньо просто надіслати HTTP-запит GET на URL BigQuery, разом із запитом потрібно передати більше інформації. У таких випадках API вимагає, щоб клієнт виконав HTTP-запит POST і надіслав JSON-запит.

Щоб змінити інформацію про набір даних, достатньо змінити атрибути локального об'єкта dsinfo та викликати метод update_dataset об'єкта клієнта, щоб переслати зміни до BigQuery:

dsinfo = bq.get_dataset(«ch05»)

print(dsinfo.description)

dsinfo.description = «Chapter 5 of BigQuery: Definitive Guide»

dsinfo = bq.update_dataset(dsinfo, ['description'])

print(dsinfo.description)

Створення порожніх таблиць. Створення порожньої таблиці виконується аналогічно до створення набору даних, і при бажанні можна ігнорувати помилку, що генерується у випадку, якщо таблиця з таким ім'ям вже існує:

table_id = '{}.ch05.temp_table'.format(PROJECT)

table = bq.create_table(table_id, exists_ok=True)

Зміна схеми таблиці. Зазвичай, рідко доводиться створювати абсолютно порожні таблиці. Найчастіше створюються таблиці зі схемою та з декількома записами. Оскільки схема є частиною набору атрибутів таблиці, схему порожньої таблиці можна змінити, як змінюють контроль доступу для набору даних. Для цього потрібно отримати об'єкт таблиці, змінити його локально, а потім відправити змінений об'єкт до BigQuery:

schema = [

bigquery.SchemaField(«chapter», «INTEGER», mode=«REQUIRED»),

bigquery.SchemaField(«title», « STRING», mode=«REQUIRED»),

]

table_id = '{}.ch05.temp_table'.format(PROJECT)

table = bq.get_table(table_id)

print(table.etag)

table.schema = schema

table = bq.update_table (table, [«schema»])

print(table.schema)

print(table.etag)

Щоб запобігти постійній гонитві за ресурсами, BigQuery забезпечує таблицю тегом при кожному оновленні. Тобто, метод get_table повертає об'єкт таблиці з атрибутом etag. При спробі вивантажити змінену схему з

використанням update_table ця спроба буде успішною тільки якщо атрибут etag зміненої таблиці збігається з атрибутом etag таблиці на сервері.

Об'єкт таблиці, що повертається, матиме нове значення в атрибуті etag. Цю поведінку можна змінити та примусово оновити схему, надавши атрибуту table. etag значення None. Якщо таблиця порожня, схему можна змінити як завгодно [34].

Але коли у таблиці є дані, будь-які зміни схеми мають бути сумісні з наявними даними у таблиці. Ви можете додати нові поля (якщо вони визначені як NULLABLE) та послабити обмеження REQUIRED до NULLABLE. Після виконання попереднього фрагмента можна відкрити веб-інтерфейс BigQuery та переконатися, що новостворена таблиця має правильну схему.

Створення порожній таблиці зі схемою. Замість створення таблиці і потім змінювати її схему, можна просто передати схему при створенні таблиці:

schema = [

bigquery.SchemaField(«chapter», «INTEGER», mode=«REQUIRED»),

bigquery.SchemaField(«title», « STRING», mode=«REQUIRED»),

]

table_id = '{}.ch05.temp_table2'.format(PROJECT)

table = bigquery.Table(table_id, schema)

table = bq.create_table(table, exists_ok=True)

print('{} created on {}'.format(table.table_id, table.created))

print(table.schema)

Таблиця, що вийшла в результаті, матиме необхідну схему:

temp_table2 created on 2019-03-03 19:30:18.324000+00 :00

[SchemaField('chapter', 'INTEGER', 'REQUIRED', None, ()),

SchemaField('title',

'STRING', 'REQUIRED', None, ())]

Створена таблиця не містить записів, та цей прийом зручно використовувати, коли передбачається виконати потокову вставку записів таблицю. Якщо нам потрібно створити таблицю з даними ініціалізувавши її

вмістом цього файлу, то потрібно використовувати завдання завантаження. Крім того, на відміну від потокової вставки, BigQuery не стягує плати за завантаження. Клієнт BigQuery на Python підтримує три методи завантаження даних: з DataFrame бібліотеки pandas, з URI та з локального файлу. Розглянемо всі три методи по порядку.

Таким чином, основними функціями та можливостями Google Big Query є:

- Управління даними – сервіс дозволяє створювати та видаляти таблиці та функції користувача, а також імпортувати дані у форматах JSON, Avro, Parquet або CSV. Щоб використовувати дані в Big Query, їх потрібно завантажити до сервісу Google Storage, а вже звідти провести імпорт даних через API. Також підтримується прямий імпорт та стримінг даних із Google Analytics.
- Запити – запити Google BigQuery створюються через стандартний діалект SQL, а результат повертається в JSON-форматі. Стандартний розмір відповіді становить 128 Мб, але також він може бути і більшим (межа необмежена) при виставленні відповідних налаштувань.
- Контроль доступу – користувачі сервісу можуть надавати стороннім особам публічний або обмежений доступ до своїх даних.
- Машинне навчання – сервіс дозволяє створювати та запускати ML-моделі за допомогою SQL-запитів.
- Інтеграції – сервіс можна використовувати як скрипт Google Apps Scripts або ж створений будь-якою іншою мовою, сумісною з REST API.

3. ОЦІНКА МОЖЛИВОСТЕЙ GOOGLE BIGQUERY ТА MYSQL

3.1. Порівняння переваг та недоліків досліджуваних баз даних

Аналізуючи переваги BigQuery слід відмітити, що BigQuery – хмарний сервіс з високою швидкістю обробки великих масивів даних. Наступна перевага – простота використання. У будь-якій іншій системі управління базами даних (СУБД) крім знання SQL доведеться довго розбиратися з тонкощами адміністрування та налаштуваннями бази [35].

- BigQuery всю адміністративну частину на себе взяв Google. У цьому сервісі немає жодних налаштувань, індексів, движків таблиць, тайм-аутів чи зовнішніх ключів. Реалізовано підтримку лише одного кодування UTF-8. Для роботи з BigQuery достатньо знати, як завантажити дані в BigQuery, і мати базові знання SQL.
- Доступність. Вартість використання Google BigQuery залежить від обсягу завантажених у нього даних і становить 5$ за 1 Тб, що набагато дешевше за оренду сервера. Після реєстрації користувач отримує $300 кредитних коштів, що діють протягом 1 року. Таким чином, протягом першого року можна скористатися сервісом абсолютно безкоштовно.
- Незважаючи на простоту, у BigQuery реалізована підтримка практично всіх функцій СУБД: віконні функції; зберігання даних як структур (нереляційні можливості); уявлення та табличні вирази (common table expression). BigQuery вміє перемикатися між стандартним SQL та діалектами. DML-операції INSERT, UPDATE та DELETE на даний момент підтримуються лише при використанні стандартного SQL. Ще одна відмінність між цими діалектами – спосіб вертикального об'єднання таблиць. У стандартному SQL для цього служить оператор UNION та ключове слово ALL або DISTINCT.
- BigQuery дозволяє використовувати такі важливі особливості, як властивості ACID (Atomicity, Consistency, Isolation, Durability - атомарність, узгодженість, ізольованість, довговічність) транзакцій, а

також автоматичну оптимізацію, завдяки чому користувачам не потрібно керувати файлами.

Спочатку служба BigQuery зберігала свій зв'язок із Dremel і була орієнтована на сканування журналів. Однак у міру збільшення кількості клієнтів, які бажали створювати сховища даних і виконувати більш складні запити, BigQuery була додана покращену підтримку з'єднань і розширені можливості SQL, включаючи аналітичні функції. У 2016 році Google додала підтримку стандартного SQL у BigQuery, що дозволило користувачам виконувати запити, використовуючи стандартну мову SQL замість незручного діалекту «DremeISQL», який спочатку використовувався.

Якщо потрібно отримати дані з Google BigQuery в електронній таблиці або BI-системі, яка з коробки не підтримує інтеграцію, то використовують Simba Drivers. Цей драйвер підтримує всі необхідні можливості, включаючи перемикання SQL діалектів.

Можливості Google BigQuery можна розширити за допомогою ряду сторонніх інструментів. Наприклад, інтегрувавши його з Google Таблиці, Microsoft Excel, QlikView, BIME Analytics та Microsoft Power BI.

Microsoft Power BI є потужним професійним сервісом для візуалізації даних, інтеграція з яким значно збільшує можливості Google BigQuery. Інтегрувати їх можна за допомогою стандартного конектора «з коробки», проте його можливості обмежені. Краще використовувати для цього безкоштовний драйвер Simba Drivers, який також підходить для зв'язування BigQuery з електронними таблицями. Крім того, підключити Microsoft Power BI можна за допомогою R-конектора, попередньо встановивши середовище розробки RStudio.

Серед недоліків сервісу BigQuery як СУБД можна виокремити неможливість підтримання: рекурсивних запитів; створення збережених процедур та функцій; транзакцій. До недавнього часу в BigQuery були відсутні особливості, властиві сховищам даних, такі як мова визначення даних (Data

Definition Language, DDL; наприклад, оператор create) та мова маніпулювання даними (Data Manipulation Language, DML; наприклад, оператор INSERT).

Крім того, недоліком є випадки значного зменшення швидкості роботи BigQuery. Адже виграш швидкості є відчутним при агрегації даних у тисячі і більше рядків, де рядкова СУБД може зависати.

Проведемо опис переваг MySQL (SQL). MySQL - реляційна база даних SQL. Перелічимо такі основні переваги MySQL, як сумісність, адже MySQL надзвичайно зріла база даних, що представляє величезне співтовариство, велике тестування і чималу стабільність.

- Сумісність: MySQL доступний для всіх основних платформ, включаючи Linux, Windows, Mac, BSD і Solaris. Він також має коннектори з такими мовами, як Node. js, Ruby, C #, C ++, Java, Perl, Python і PHP, що означає, що він не обмежується мовою запитів SQL.
- Економічно ефективний: містить безкоштовні бази даних з відкритим вихідним кодом.
- Репліцирована: база даних MySQL може бути репліцирована на кілька вузлів і це означає, що робоче навантаження може бути зменшено, а масштабованість і доступність додатку може бути збільшено.
- Подільність: хоча поділ неможливо в більшості баз даних SQL, це можливо зробити на серверах MySQL, що є вигідним.

Оцінюючи функціональні переваги MySQL можна відзначити такі:

1. Використовуваний основний потік є багатопоточним і підтримує кілька процесорів.
2. Існує кілька типів стовпців: 1, 2, 3, 4 та 8-байтове ціле число без знака / зі знаком, FLOAT, DOUBLE, CHAR, VARCHAR, TEXT, BLOB, DATE, TIME, DATETIME, TIMESTAMP, YEAR та Тип ENUM .
3. Він реалізує бібліотеку функцій SQL через високооптимізовану бібліотеку класів і працює настільки швидко, наскільки це можливо, зазвичай після ініціалізації запиту не повинно бути виділення пам'яті. Жодних витоків пам'яті.

4. Він повністю підтримує пропозиції SQL GROUP BY та ORDER BY та підтримує агреговані функції (COUNT(), COUNT(DISTINCT), AVG(), STD(), SUM(), MAX() та MIN()). Можна змішувати таблиці різних баз даних в одному запиті.
5. Підтримка LEFT 0UTER JOIN та ODBC ANSI SQL.
6. Усі стовпці мають значення за промовчанням. Можна використовувати INSERT для вставки підмножини стовпця таблиці, і для тих стовпців, які не потрібно вказувати явно, встановлюються стандартні значення.
7. MySQL може працювати на різних платформах. Підтримка C, C++, Java, Perl, PHP, Python та TCL API.

Система MySQL має такі обмеження у своєму функціоналі, що не дозволяє використовувати її для роботи з програмами:

– Недостатня надійність. У питаннях надійності деяких процесів роботи з даними (наприклад, зв'язок, транзакції, аудит) MySQL поступається деяким іншим СУБД.
– Низька швидкість розробки. Як і багатьом іншим програмним продуктам з відкритим кодом, MySQL не вистачає деякої технічної досконалості, що часом позначається на ефективності процесів розробки.

Оцінюючи функціональні недоліки MySQL можна відзначити такі:

1. Найбільшим недоліком MySQL є його система безпеки, яка в основному складна, а не стандартна. Крім того, вона змінюється лише тоді, коли mysqladmin викликається для перечитування дозволів користувачів.
2. Одним із інших серйозних недоліків MySQL є відсутність стандартного механізму RI (Referential Integrity-RI); відсутність обмеження RI (обмеження фіксованого діапазону у заданому домені поля) може бути досягнуто за допомогою великої кількості типів даних. скласти.
3. MySQL не має мови збережених процедур, що є найбільшим обмеженням для програмістів, що звикли до баз даних корпоративного рівня.
4. MySQL не підтримує гаряче резервне копіювання.

5. Ціна MySQL залежить від платформи та методу встановлення. Linux MySQL є безкоштовним, якщо його встановлено користувачем або системним адміністратором, а не третьою особою, а третій план повинен сплачувати ліцензійний збір. Самостійне встановлення Unix або Linux безкоштовне, стороннє встановлення Unix або Linux коштує 200 доларів.

В цілому, система MySQL не підходить для використання, якщо:

– необхідно відповідати стандарту SQL, який дана система підтримує лише частково;
– проєкт передбачає багатопоточність даних, оскільки при здійсненні паралельних операцій читання/запису MySQL можуть виникати проблеми;
– існуючий функціонал MySQL не здатний забезпечити весь набір можливостей роботи з базою даних.

3.2. Перспектива використання Google BigQuery

Перспективність використання Google BigQuery полягає у розширенні можливостей сумісного використання даної бази даних з іншими програмними продуктами та оптимізація продуктивності запитів.

Час, що витрачається на виконання запиту, залежить від обсягу даних, які витягуються з сховища, організації цих даних, кількості етапів, необхідні обробки запиту, можливості розпаралелювання цих етапів, обсягу даних, оброблюваних кожним етапом, і обчислювальної дорожнечі кожного з етапів. Загалом простий запит, який читає три стовпці, буде виконуватися на 50% довше запиту, який читає лише два стовпці, тому що запит із трьома стовпцями повинен прочитати на 50% більше даних. Запит, що включає угруповання, зазвичай виконується повільніше, ніж запит без угруповання, тому що операція угруповання додає додатковий етап обробки запиту [11].

Оскільки BigQuery має REST API, то для оцінки часу виконання запитів можна використовувати будь-який інструмент роботи з веб-службами. Часто виникає потреба виміряти швидкість виконання сервера, на якому ці

інструменти відсутні. У таких випадках можна вдатися до Unix-утилітів time та curl. Код SQL і JSON можна помістити в змінні Bash:

```
read -d QUERY_TEXT << EOF
SELECT \`.london_bicycles.cycle_hire GROUP BY
start_station_name
, AVG(duration) as duration
, COUNT(duration) as num_trips
FROM \`bigquery-public-data\`.london_bicycles.cycle_hire
GROUP BY start_station_name
ORDER BY num_trips DESC
LIMIT 5 EOF
read -d '' request << EOF
{
«useLegacySql»: false,
«useQueryCache»: false,
«query»: \»${QUERY_TEXT}\»
}
 EOF
 request=$(echo «$request» | tr '\n' ' ')
```

Використовують інструмент командного рядка gcloud для отримання токена доступу та ідентифікатора проєкту, що необхідні для звернення до REST API:

```
access_token=$(gcloud auth application-default print-access-token)
PROJECT=$(gcloud config get-value project)
```

Тепер, коли всі готово, можна виконати запит кілька разів і отримати загальний час, щоб розрахувати середню продуктивність запиту і зменшити вплив на оцінку випадкових мережевих збоїв:

```
NUM_TIMES=10
time for i in $(seq 1 $NUM_TIMES); do
echo -en «\r ... $i / $NUM_NUMTIMES ...»
```

*curl --silent *

*-H «Authorization: Bearer $access_token» *

*-H «Content-Type: application/json» *

*-X POST *

*-d «$request» *

«https://www.googleapis.com/bigquery/v2/projects/$PROJECT/queries» > /dev/null done

При цьому отримаємо наступні результати:

Real 0m16.875s

User 0m0.265s

Sys 0m0 .109s

Загальний час виконання запиту 10 разів склав 16.875 секунд, тобто в середньому на виконання одного запиту йшло 1.7 секунд.

Розглянемо можливість сумісної роботи досліджуваних систем. Оскільки BigQuery виступає як єдине місце збереження необроблених даних, то MySQL може виступати як шар кеша поверх нього та зберігати лише невеликі агреговані таблиці та надавати бажану відповідь за запитом. Проаналізуємо черговість дій при імпорті із MySQL в Big Query.

Крок 1: експорт з MySQL

Конфігурація:

Польовий роздільник: символ керування 001

Encloser: "(none)

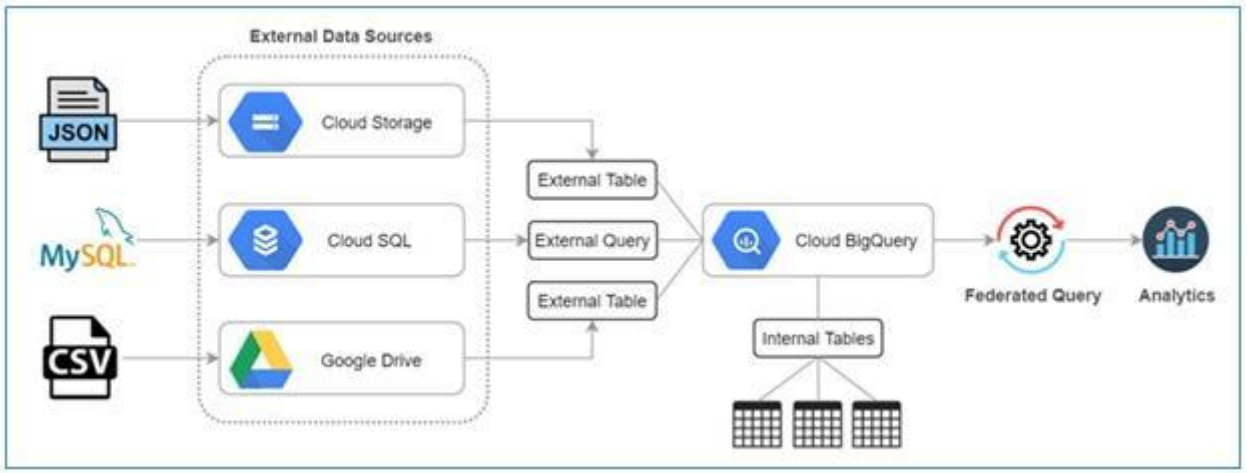


Рисунок 3.1. Схематичне зображення процесу імпорту із MySQL в Big Query

Запит із MySQL представлено нижче за допомогою AWS RDS Aurora, і тому синтаксис трохи відрізняється від стандартного MySQL (файл записується на S3):

*SELECT * FROM my_table*

INTO OUTFILE S3 's3://xxxxx/tmp/my_table/data'

CHARACTER SET UTF8MB4

FIELDS TERMINATED BY x'01'

OPTIONALLY ENCLOSED BY ''

MANIFEST OFF

OVERWRITE ON

Крок 2. Копіюємо набір даних у хмарне сховище за допомогою gsutil

gsutil rsync -m s3://xxxxx/tmp/my_table/ gs://xxxxx/tmp/my_table/

Крок 3: Завантаження даних у Big Query за допомогою CLI

bq load --source_format=CSV --field_delimiter=^A --null_marker=«\N» --quote=«« project:base.my_table gs://xxxxx/tmp/my_table/ ./schema.json*

Примітка:

^ A - уявлення керуючого символу, що створений в Windows за допомогою Alt + 001, на linux-оболонці з Ctrl + V і Ctrl + A.

Google BigQuery може представляти майбутнє аналітики великих даних завдяки вищезгаданій архітектурі, яка забезпечує оптимальну продуктивність без необхідності керування інфраструктурою чи повторної побудови індексів.

Різні впливові компанії, підприємці та ентузіасти Google позитивно використовують можливості Google BigQuery та високо оцінили його функціональність та внесок у керування їхнім бізнесом. Наприклад, Twitter повідомив, що за допомогою BigQuery вони змогли демократизувати свій аналіз даних та поділитися інформацією про компанію із широким колом внутрішніх груп. Група Alpega також спромоглася оптимізувати свої інновації за допомогою системи, використовуючи аналітику в реальному часі, яку їм раніше не вдавалося отримати.

ВИСНОВКИ

1. Розглянуто існуючі на даний час моделі зберігання даних та сказано, що найбільш розповсюдженими моделями є постреляційна, багатомірна, об'єктно-орієнтована. Для кожної моделі дано короткий опис та дано схематичне зображення.

2. Для динамічної моделі бази даних зроблено математичний опис і для реляційних баз дано поглиблене визначення та опис. Встановлено, що реляційні бази даних надійні, а транзакції завжди виконуються, як очікується, завдяки принципам ACID. Наголошено, що Перспективи розвитку архітектур СКБД пов'язані з розвитком концепції обробки нетрадиційних даних та їх інтеграцією, обміном даними з різних СКБД, багатокористувацької технології в локальних мережах.

3. В практичній частині дослідження проведено аналіз функціонального інструментарію MySQL та Google BigQuery. MySQL є рішенням для малих і середніх додатків, а Google BigQuery використовується для великих хмарних баз даних. Для досліджуваних систем дано їх математичний опис та наведено схематичну архітектуру, вказано переваги та недоліки, наголошено на областях практичного використання.

4. Окремо проведено порівняння досліджуваних систем та вказано можливий шлях імпорту даних із MySQL до Google BigQuery. Зроблено висновок про те, що можливості Google BigQuery можна розширити за допомогою ряду сторонніх інструментів. Наприклад, інтегрувавши його з Google Таблиці, Microsoft Excel, QlikView, BIME Analytics та Microsoft Power BI.

5. Встановлено, що перспективність використання Google BigQuery полягає у розширенні можливостей сумісного використання даної бази даних з іншими програмними продуктами та оптимізація продуктивності запитів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Алгоритми та структури даних / уклад. О. В. Щербаков, Ю. Е. Парфьонов, В. М. Федорченко. Харків : ХНЕУ ім. С. Кузнеця, 2017. 58 с.
2. Бобрешов-Шишов Д. І. Динамічне керування структурою розподіленої бази даних. *Молодий вчений*. 2015. № 7. С. 51–53.
3. Богач І. В., Довгалець С. М., Дубовой В. М., Алгоритми розв’язання задач з програмування. Решебник. Вінниця: ВНТУ, 2017 119 с.
4. Борис Т. В. Управління великими масивами даних в якості послуги. *Проблеми телекомунікацій*. К., 2014. С. 243–245.
5. Васильєв О. Програмування на С++ в прикладах і задачах: навч. Посібник К.: Ліра-К, 2017. 258 с.
6. Ганжела С. І. Основи інформатики з елементами програмування та сучасні інформаційні технології навчання. Кропивницький: РВВ ЦДПУ ім. В. Винниченка, 2017. 61 с.
7. Додонов А. Г., Ландэ Д. В. Розпізнавання інформаційних операцій. К.: ООО «Инжиниринг», 2017. 284 с.
8. Кузь М. В., Соловко Я. Т. Методологія формування узагальненого критерію якості програмного забезпечення в умовах невизначеності. *Вісник Вінницького політехнічного інституту*. 2015. №5. С. 104–107.
9. Ліщинська Л. Б. Основні аспекти автоматизації роботи з клієнтами засобами CRM-систем. *Вісник Хмельницького національного університету. Економічні науки.* 2015. № 5(1). С. 206–209.
10. Матвієнко М. П. Алгоритми та структури даних : навч. посіб. / М. П. Матвієнко. К.: Видавництво «Ліра-К», 2014. 340 с.
11. Методи та моделі розроблення комп’ютерних систем і мереж : монографія / В. С.Пономаренко, С. В. Мінухін. Х.: Вид. ХНЕУ, 2016. 316 с.
12. Милорадов К. А. Підвищення ефективності взаємодії з клієнтами за допомогою хмарних сервісів. *Міжнародний журнал експериментальної освіти.* 2015. № 3-1. С. 70-71.
13. Мулеса О. Ю. Основи мови запитів SQL. Ужгород, 2015. 48 с.

14. Оліфер В. Г. Мережеві операційні системи. СПб.: Пітер, 2016. 544 с.
15. Панфілов К. В. Аналіз систем моніторингу мережевого обладнання мережі передачі даних. Самара : ПГУТИ, 2019. 96 с.
16. Пекарський Б. Г. Основи програмування: навчальний посібник. К.: Кондор, 2018. 270 с.
17. Фісун М., Дворецький М., Дворецька С. Побудова моделей для оптимізації структури бази даних вузла у корпоративних інформаційних системах. *ІТКІ,* 2020. vol 48, № 2, С. 52-60.
18. Ярцев В.П. Розподілені бази даних: навчальний посібник. К. ДУТ 2018. 97 с.
19. Шиндер Л. Д. Основи комп'ютерних мереж / Л. Д. Шиндер. М.: 2015. 152 с.
20. Яргер, Р. Дж.; Риз, Дж.; Кинг, Т. MySQL и mSQL: Базы данных для небольших предприятий и Интернета; СПб: Символ-Плюс, 2013. 560 с.
21. Abualigah, L.; Gandomi, A.H. 2021. Advances in Meta-Heuristic Optimization Algorithms in Big Data Text Clustering. Electronics. № 10, 101.
22. Acharjya DP, Ahmed K. 2016. A survey on big data analytics: challenges, open research issues and tools. *International Journal of Advanced Computer Science and Applications.* 7(2):511–518.
23. Ahmed I, Ahmad M, Jeon G, Piccialli F. A framework for pandemic prediction using big data analytics. Big Data Research 100190. Epub ahead of print Jan 16 2021 DOI 10.1016/j.bdr.2021.100190.
24. Alkurd R, Abualhaol I, Yanikomeroglu H. 2020. Big-data-driven and AI-based framework to enable personalization in wireless networks. *IEEE Communications Magazine*. 58(3):18–24.
25. Altman, M., Wood, A., O'Brien, D.R» and Gasser, U. (2018). Practical Approaches to Big Data Privacy over Time. International Data Privacy Law. doi:10.1093/idpl/ipx027.

26. AlZubi AA. 2020. Big data analytic diabetics using map reduce and classification techniques. *The Journal of Supercomputing*. 76(6):4328–4337 DOI 10.1007/s11227-018-2362-1.
27. Asencio-Cortés G, Morales-Esteban A, Shang X, Martínez-Álvarez F. 2018. Earthquake prediction in California using regression algorithms and cloud-based big data infrastructure. *Computers & Geosciences*. 115:198–210 DOI 10.1016/j.cageo.2017.10.011.
28. Borodo SM, Shamsuddin SM, Hasan S. 2016. Big data platforms and techniques. Indonesian. *Journal of Electrical Engineering and Computer Science*. 1(1):191–200 DOI 10.11591/ijeecs.v1.i1.pp191-200.
29. Ding W, Zhao Z, Wang J, Li H. 2020. Task allocation in hybrid big data analytics for urban IoT applications. ACM Transactions on Data Science 1(3):1–22.
30. Haleem A, Javaid M, Khan IH, Vaishya R. Significant Applications of Big Data in COVID-19 Pandemic. *Indian J Orthop [Internet]*. 2020;54(4):526–8.
31. Hook D. W., Porter S. J. 2021. Scaling Scientometrics: Dimensions on Google BigQuery as an Infrastructure for Large-Scale Analysis. *Front. Res. Metr. Anal.* **6**:656233. doi: 10.3389/frma.2021.656233.
32. Favaretto, M., De Clercq, E., Schneble, C. O., & Elger, B. S. (2020). What is your definition of Big Data? Researchers' understanding of the phenomenon of the decade. *PloS one*, 15(2), e0228987. https://doi.org/10.1371/journal.pone.0228987.
33. Feng M, Zheng J, Ren J, Hussain A, Li X, Xi Y, Liu Q. 2019. Big data analytics and mining for effective visualization and trends forecasting of crime data. *IEEE Access*. 7:106111–106123 DOI 10.1109/ACCESS.2019.2930410.
34. Furht B, Villanustre F. 2016. Introduction to big data. In: Big data technologies and applications. Berlin, Heidelberg: Springer, 3–11.
35. Gandomi, A.H.; Chen, F.; Abualigah, L. Machine Learning Technologies for Big Data Analytics. Electronics 2022, 11, 421. https:// doi.org/10.3390/electronics11030421.

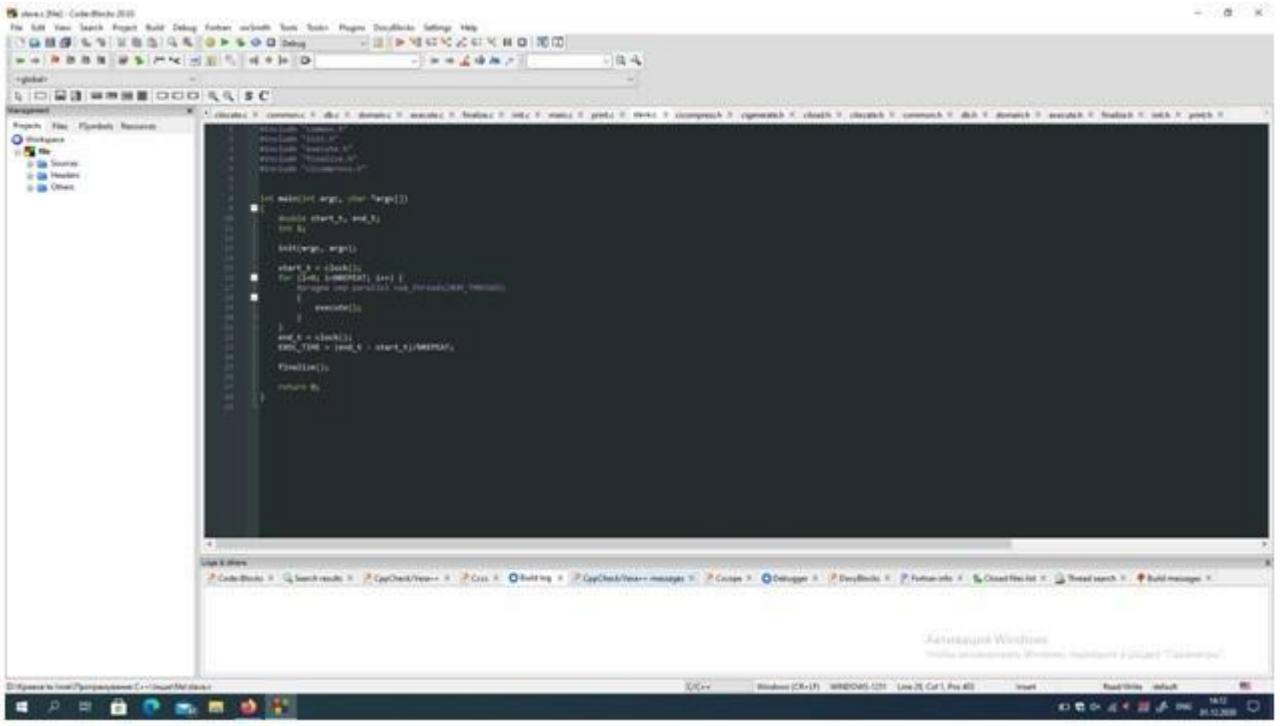
36. Gantz, J., & Reinsel, D. (2012). The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east. IDC iView: *IDC Analyze the future,* (2012), 1-16.
37. Győrödi CA, Dumşe-Burescu DV, Zmaranda DR, Győrödi RŞ, Gabor GA, Pecherle GD. Performance Analysis of NoSQL and Relational Databases with CouchDB and MySQL for Application's Data Storage. ***Applied Sciences***. 2020; 10(23):8524. https://doi.org/10.3390/app10238524.
38. Kannan N, Sivasubramanian S, Kaliappan M, Vimal S, Suresh A. 2019. Predictive big data analytic on demonetization data using support vector machine. *Cluster Computing*. 22(6):14709–14720 DOI 10.1007/s10586-018-2384-8.
39. Kaur P, Sharma M, Mittal M. 2018. Big data and machine learning based secure healthcare framework. *Procedia Computer Science.* 132:1049–1059 DOI 10.1016/j.procs.2018.05.020.
40. Klein S. 2017. The world of big data and IoT. In: IoT solutions in Microsoft's azure IoT suite. Berkeley: Apress, 3–13.
41. Krocz, K., Kizun, O., & Skublewska-Paszkowska, M. (2020). Perfomance analysis of relational databases MySQL, PostgreSQL, MariaDB and H2. *Journal of Computer Sciences Institute*, *14*, 1-7. https://doi.org/10.35784/jcsi.1565.
42. Lu Q, Li Z, Zhang W, Yang LT. 2017. Autonomic deployment decision making for big data analytics applications in the cloud. Soft Computing 21(16):4501–4512 DOI 10.1007/s00500-015-1945-5.
43. Manyika J. Big Data: The next frontier for innovation, competition, and productivity. *McKinsey Global Institute*, June, 2011. McKinsey, 2011. P. 27–31.
44. Mazumder S. 2016. Big data tools, platforms. In: Big data concepts, and theories, and applications. Cham: Springer, 29–128.

45. Mittal S, Sangwan OP. 2019. Big data analytics using machine learning techniques. In: 2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence). Piscataway: IEEE.
46. MySQL is the Database of the Year. https://db-engines.com/en/blog_post/83. January 2020.
47. Nicolalde FC, Silva F, Herrera B, Pereira A. 2018. Big data analytics in IOT: challenges, open research issues and tools. In: World conference on information systems and technologies. Cham: Springer.
48. Oussous A, Benjelloun F-Z, Lahcen A, Belfkih S. 2018. Big data technologies: a survey. Journal of King Saud University-Computer and Information Sciences 30(4):431–448 DOI 10.1016/j.jksuci.2017.06.001.
49. Patgiri R, Ahmed A. 2016. Big data: The v's of the game changer paradigm. In: 2016 IEEE 18th international conference on high performance computing and communications; IEEE 14th international conference on smart city; IEEE 2nd international conference on data science and systems (HPCC/SmartCity/DSS). Piscataway: IEEE.
50. Pulgar-Rubio F, Rivera-Rivas AJ, Pérez-Godoy MD, González P, Carmona CJ, del Jesus MJ. 2017. MEFASD-BD: multi-objective evolutionary fuzzy algorithm for subgroup discovery in big data environments-a mapreduce solution. Knowledge-Based Systems 117:70–78 DOI 10.1016/j.knosys.2016.08.021.
51. Yun D, Wu CQ, Rao NS, Kettimuthu R. 2019. Advising big data transfer over dedicated connections based on profiling optimization. IEEE/ACM Transactions on Networking 27(6):2280–2293.
52. Zhu C, Cheng G, Wang K. 2017. Big data analytics for program popularity prediction in broadcast TV industries. IEEE Access 5:24593–24601 DOI 10.1109/ACCESS.2017.2767104.

ДОДАТКИ

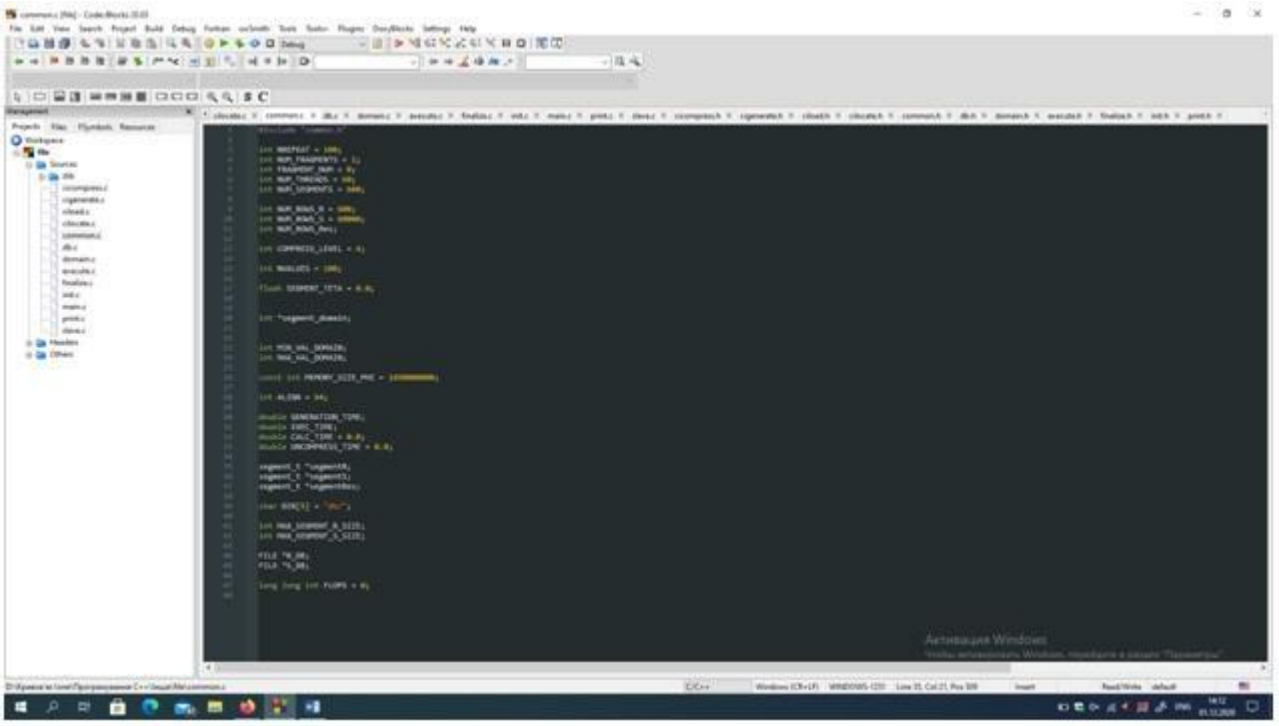
Додаток А

Робоче вікно при загрузці програми



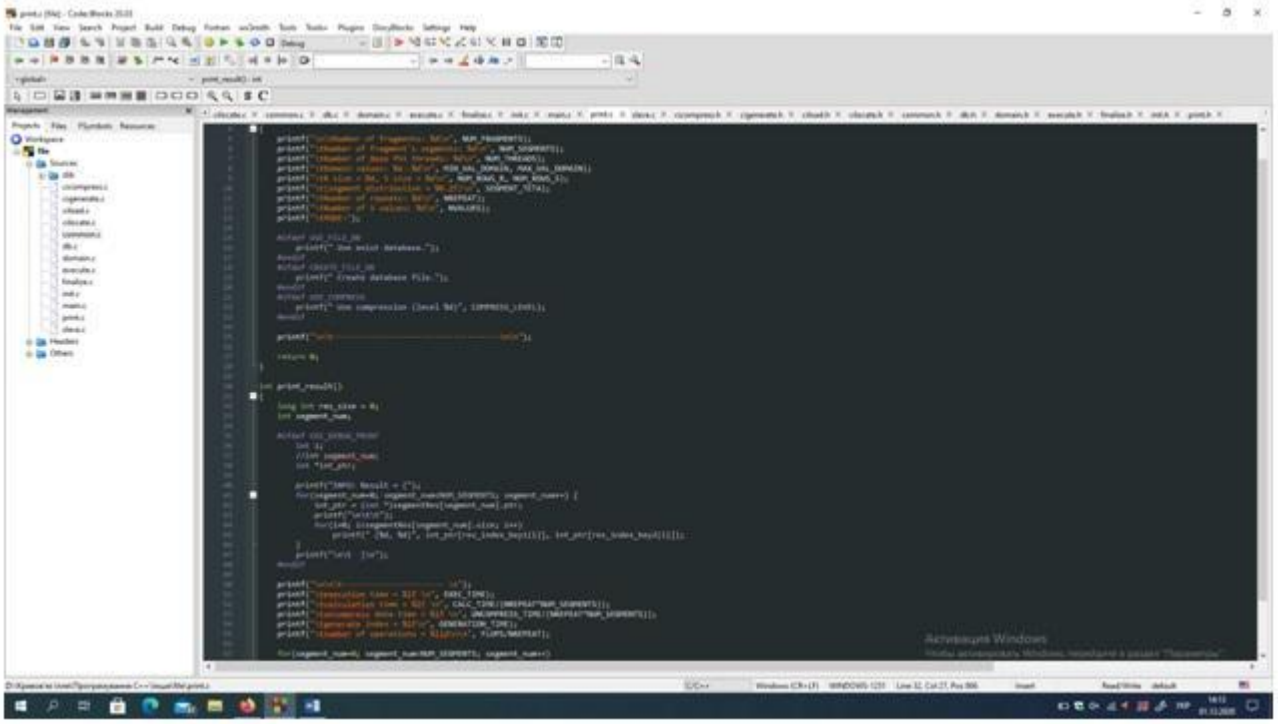
Додаток Б

Фрагмент змінних, які використовуються у програмі



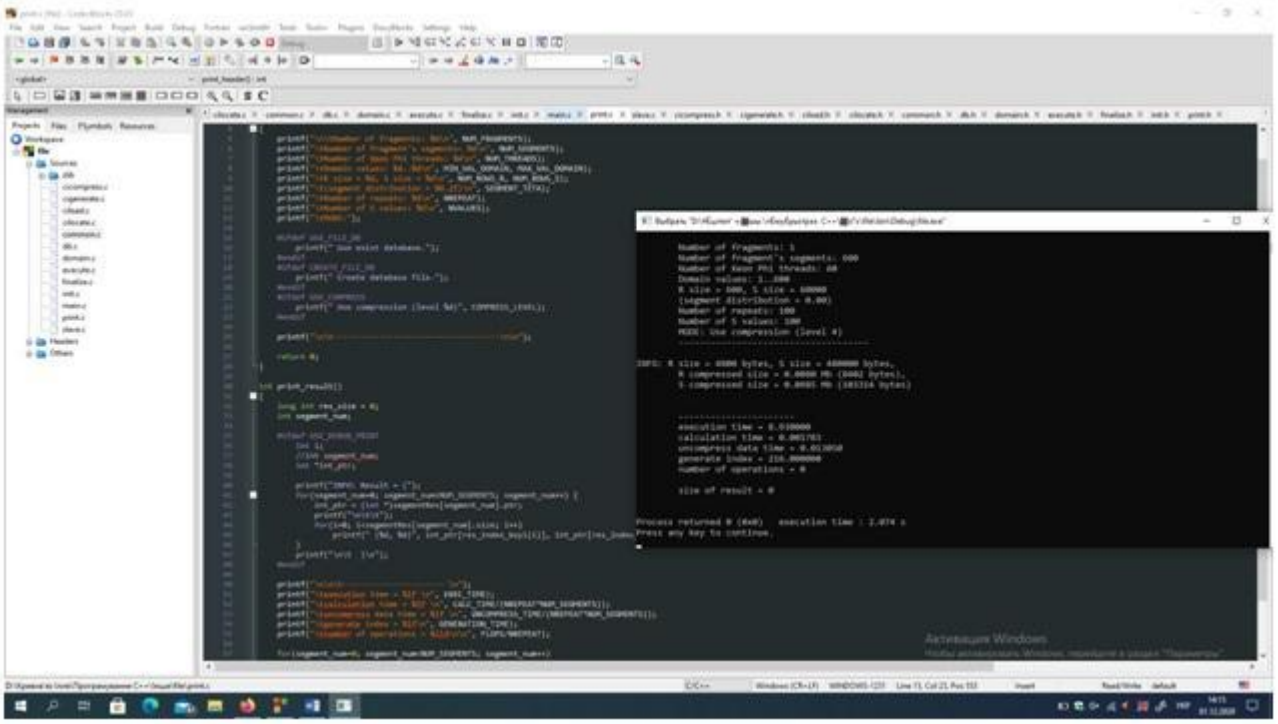
Додаток В

Констатуючі результати звернення програми до баз даних



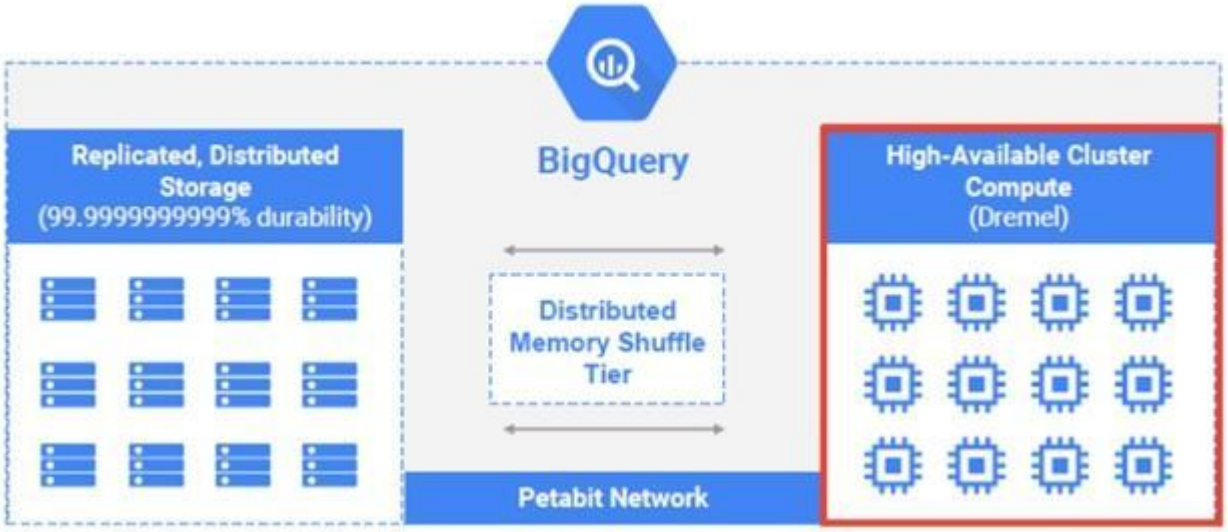
Додаток Г

Результати запитів програмою даних з бази



Додаток Д

Шляхи оптимізації запитів у Google BigQuery



Додаток Г

Презентація

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Google BigQuery як альтернатива MySQL в контексті мікросервісної архітектури

Студент: Трофименко Вадим Володимирович, ППЗМ-71
Науковий керівник: д.т.н., доц., Жебка Вікторія Вікторівна

Київ-2022

Мета, об’єкт, предмет дослідження

Мета: покращення процесу аналізу та вивчення даних за допомогою Google BigQuery як альтернативи MySQL.

Об’єкт дослідження: процес аналізу та вивчення даних.

Предмет роботи: методи та засоби роботи з Google BigQuery та MySQL.

Завдання:

1. Зробити широкий літературний пошук з детальним аналізом наукової інформації.
2. Провести систематизацію та адаптацію отриманих літературних результатів.
3. Розробити рекомендації з використання отриманих даних.

Наукове завдання:?

Математичний опис динамічної моделі бази даних

$$y_j(t) = \sum_{k=1}^{\infty} \sum_{i_1=1}^{\nu} \dots \sum_{i_k=1}^{\nu} \int_0^t \dots \int_0^t w^j{}_{i_1 i_2 \dots i_k}(\tau_1, \tau_2, \dots, \tau_k) \prod_{l=1}^{k} x_{i_l}(t-\tau_l) d\tau_l$$

$w^j_{i_1..i_k}(\tau_1,\dots\tau_k)$ – багатомірні вагові функції (БВФ) k-го порядку по $i_1,\dots, i_k$ входам та j-му виходу (j=1,2,...,μ), ν, μ – кількість входів та виходів відповідно, $x(t)$ – вхідний вплив, $y_j(t)$ – відгук об'єкта на j-му виході при нульових початкових умовах.

Експериментальні показники для оцінки якості системи класифікації

Оцінка помилок класифікації визначається виразом:

$$\delta_i = L_i / N_i,$$

де L_i – кількість об'єктів і-го класу, помилково віднесених до іншого класу k (k≠i); N_i – кількість елементів і-го класу в вибірці; i=1, 2,..., m; m – кількість класів стану ОК.

Оцінка імовірності правильного розпізнавання P, середня по всіх класах:

$$P = 1 - \sum_{i=1}^{m} L_i \cdot \left(\sum_{i=1}^{m} N_i \right)^{-1}$$

Оцінка середнього ризику R (середньої вартості прийняття рішення):

$$R = \sum_{i=1}^{m} \delta_i s_i p(\Omega_i)$$

де s_i – вартості помилок δ_i,

$$p(\Omega_i) = N_i / \sum_{j=1}^{m} N_j$$

– апріорна імовірність появи класу Ω_i.

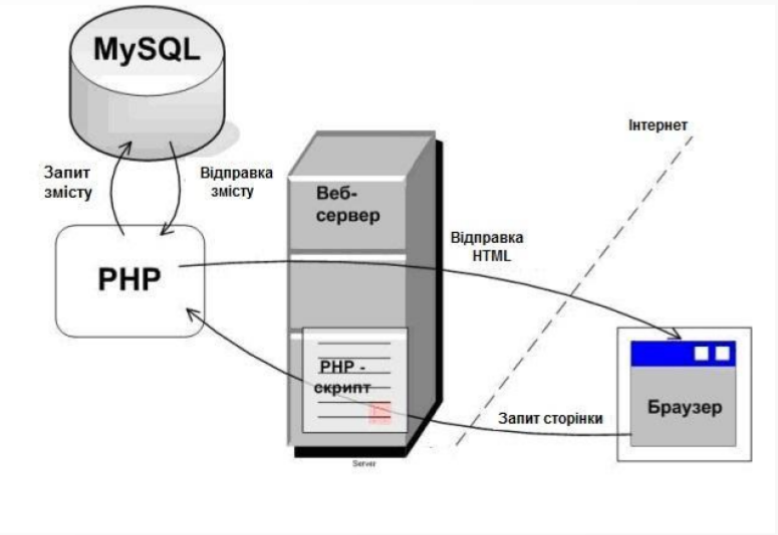
Переваги MySQL

1. Сумісність.
2. Економічна ефективність.
3. Репліцированість.
4. Подільність.

Функціональні переваги

1. Використовуваний основний потік є багатопоточним і підтримує кілька процесорів.
2. Існує кілька типів стовпців.
3. Реалізує бібліотеку функцій SQL через високооптимізовану бібліотеку класів.
4. Він повністю підтримує пропозиції SQL GROUP BY та ORDER BY та підтримує агреговані функції.
5. Підтримка LEFT 0UTER JOIN та ODBC ANSI SQL.
6. Усі стовпці мають значення за промовчанням.
7. MySQL може працювати на різних платформах.

Схема обробки запитів СУБД MySQL



Недоліки MySQL

1. Недостатня надійність.
2. Низька швидкість розробки.

Функціональні недоліки

1. Система безпеки, яка в основному складна, а не стандартна.
2. Відсутність стандартного механізму RI (Referential Integrity-RI)
3. Не має мови збережених процедур.
4. Не підтримує гаряче резервне копіювання.
5. Ціна MySQL залежить від платформи та методу встановлення.

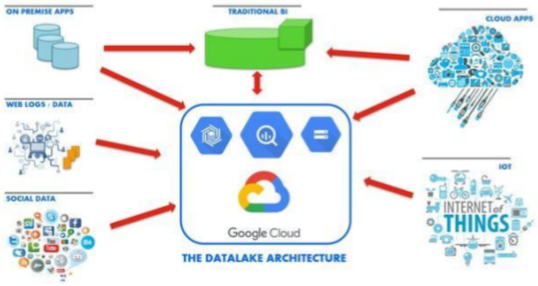
Переваги Google BigQuery

1. BigQuery всю адміністративну частину на себе взяв Google.
2. Доступність.
3. У BigQuery реалізована підтримка практично всіх функцій СУБД.
4. Можливість використовувати ACID.
5. Розширення за допомогою інших інструментів.

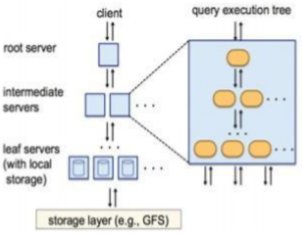
Недоліки Google BigQuery

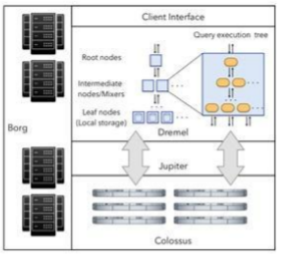
1. Неможливість підтримки рекурсивних запитів.
2. Неможливість створення збережених процедур та функцій.
3. Виграш у швидкості присутній лише при обробці тисячі і більше рядків.

Google Cloud екосистема та архітектура Google BigQuery



Google Cloud екосистема





архітектура Google BigQuery

Черговість дій при імпорті із MySQL в Big Query

Крок 1: експорт з MySQL

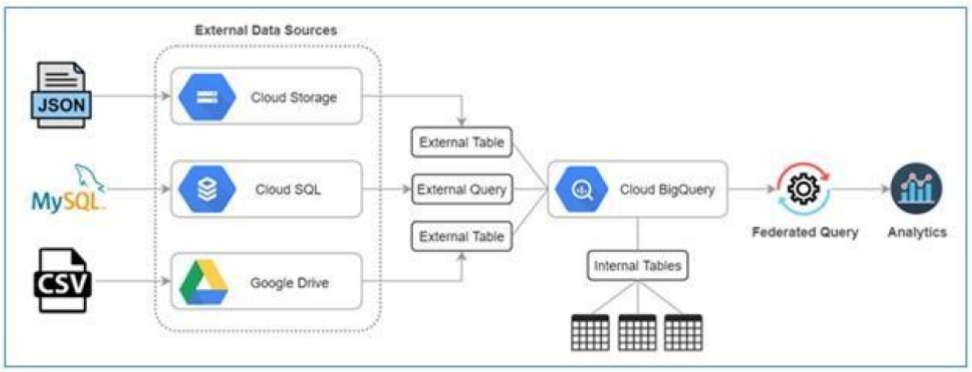
*SELECT * FROM my_table*
INTO OUTFILE S3
's3://xxxxx/tmp/my_table/data'
CHARACTER SET UTF8MB4
FIELDS TERMINATED BY x'01'
OPTIONALLY ENCLOSED BY ''
MANIFEST OFF
OVERWRITE ON

Крок 2: Копіюємо набір даних у хмарне сховище за допомогою gsutil gsutil rsync -m s3://xxxxx/tmp/my_table/ gs://xxxxx/tmp/my_table/

Крок 3: Завантаження даних у Big Query за допомогою CLI

bq load --source_format=CSV
--field_delimiter=^A --null_marker=«\N»
--quote=«« project:base.my_table
gs://xxxxx/tmp/my_table/ ./schema.json*

Схематичне зображення процесу імпорту із MySQL в Big Query



Апробація результатів дослідження

Стаття

Трофименко В.В., Жебка В.В., Корецька В.О. GOOGLE BIGQUERY ЯК АЛЬТЕРНАТИВА MYSQL. Зв'язок. 2022. № 2 (156). С.20-28

Тези доповіді

Трофименко В.В. База даних Google BigQuery як майбутнє аналітики великих даних. Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в ІКТ». Збірник тез. – К.: ДУТ, 2022.

Висновки

- Розглянуто існуючі на даний час моделі зберігання даних та сказано, що найбільш розповсюдженими моделями є постреляційна, багатомірна, об'єктно-орієнтована.
- Для динамічної моделі бази даних зроблено математичний опис і для реляційних баз дано поглиблене визначення та опис.
- Проведено аналіз функціонального інструментарію MySQL та Google BigQuery. MySQL є рішенням для малих і середніх додатків, а Google BigQuery використовується для великих хмарних баз даних.
- Окремо проведено порівняння досліджуваних систем та вказано можливий шлях імпорту даних із MySQL до Google BigQuery.
- Встановлено, що перспективність використання Google BigQuery полягає у розширенні можливостей сумісного використання даної бази даних з іншими програмними продуктами та оптимізація продуктивності запитів.

Дякую за увагу!