УДК 004.056.5

УКПП ХХХХХ

№ держреєстрації ХХХХХ

Інв. № ХХХХХХ

**Національна академія наук України**

Харківський національний університет радіоелектроніки

61166, м. Харків, пр. Науки, 14, тел. (057)7021013

ЗАТВЕРДЖУЮ

Ректор Харківського національного

університету радіоелектроніки

\_\_\_\_\_\_\_\_\_\_\_\_\_ (ініціали, прізвище)

(підпис)

(дата)

ЗВІТ

ПРО НАУКОВО-ДОСЛІДНУ РОБОТУ

ДОСЛІДЖЕННЯ МЕТОДІВ

ЗБЕРЕЖЕННЯ ВЕЛИКИХ ДАНИХ

Науковий керівник НДР

канд. техн. наук,

доц. на кафедрі 503 Піскачов О.І. (ініціали, прізвище)

2016

**СПИСОК АВТОРІВ**

|  |  |  |
| --- | --- | --- |
| Науковий керівник НДР  канд. техн. наук, доцент на кафедрі 503 | (підпис)  (дата) | О. І. Піскачов  (вступ; розділ 1, 3, висновки) |
| Відповідальний виконавець  мол. наук. співроб. | (підпис)  (дата) | Орлов С.В.  (реферат; розділ 2,4,5) |

**РЕФЕРАТ**

Звіт про НДР: 79 с., 5 ч., 18 табл., 13 рис, 4 дод., 25 джерела.

SQL, NoSQL, VISUAL STUDIO, C#, ШВИДКІСТЬ CRUD ОПЕРАЦІЙ, НАВАНТАЖЕННЯ, РЕПЛІКАЦІЯ, АГРЕГАТ, ВІДНОШЕННЯ, ТАБЛИЦЯ, НОРМАЛІЗАЦІЯ.

Об'єкт дослідження ­– процес збереження великих даних у реляційних та нереляційних базах даних та їх вплив на метрики операцій роботи з даними.

Мета роботи – дослідження моделі збереження великих даних та вибір відповідної моделі збереження даних для розробки програмних додадків на основі отриманих метрик та характристик процесів збереження даних. Проаналізувати та провести систематизацію впливу об’єму даних та відповідної моделі на швидкодію та час откліку системи.

Методи дослідження – моделювання процесу роботи з даними великого обсягу та процесу роботи з даними маючими різнй тип, структуру та моделі з використанням вибраних за результатами попередніх досліджень метрик та характеристик, математичне моделювання процесу аналізу та обробляння даних.

Методом вирішення задачі є розробка сервісу на за допомогою мови програмування C# з використання NoSQL баз даних, а також використанням реляційної бази даних MS SQL Server. Буде розроблено математичну модель, яка буде здатна проаналізувати отримані практичні розрахунки та допомогти прийняти рішення стосовно, відповісти на питання мети цього дослідження, яка модель зберігання даних у яких випадках має свої переваги та коли її застосування математично обґрунтовано.

На основі результатів виконаних досліджень проведено порівняльний аналіз моделей за їх спільними основними характеристиками (швидкість вибору даних, швидкість оновлення даних, кількість витрат пам’яті та ін.) та на його підставі розроблено веб-систему, яка використовує різні методи збереження даних та визначено кращі приклади застосування кожного з методів для вирішення промислових завдань.

Дане дослідження досить актуальне на теперішний час та має дуже велику галузь застосувань, оскільки при масштабування інформаційних ресурсів основне навантаження припадає на доступ до даних та масштабування функцій роботи з базами даних. У цьому дослідженні буде зроблено висновок про те, на які СУБД краще робити основний акцент.

**ЗМІСТ**

|  |  |
| --- | --- |
| Перелік скорочень та умовних познак.................................................................................................  Вступ……………………………………………………………………………………………….. | 5 |
| 6 |
| 1 Аналіз предметної галузі і постановка задачі ………………………....................................... | 9 |
| 2 Реляційні та нереляційні моделі збереженні даних…………………………………………... | 14 |
| 2.1 Опис реляційної та нереляційної моделі …………………………………………………... | 14 |
| 2.2 Формальний опис структури великих даних………………………………………………… | 30 |
| 2.3 Моделі асоціацій між сутностями та характеристиками для різних категорій NoSQL БД | 31 |
| 2.4 Особливості NoSQL моделей збереження даних…………………………………………… | 35 |
| 2.5 Парадигма MapReduce, BSON, JSON………………………………………………………... | 44 |
| 3 Опис методів та моделей магістерського дослідження……………………………………….. | 48 |
| 3.1 Методи магістерського дослідження…………………………………………………………. | 48 |
| 3.2 Модель магістерського дослідження…………………………………………………………. | 50 |
| 4 Програмна реалізація……………………………………………………………………………. | 55 |
| 4.1 Реляційна MS SQL база даних………………………………………………………………... | 55 |
| 4.2 Сховище ключ-значення Redis………………………………………………………………... | 59 |
| 4.3 Розподілене сховище Cassandra……………………………………………………………… | 60 |
| 5 Проведення експерименту з порівняння баз даних…………………………………………… | 63 |
| Висновки…………………………………………………………………………………………… | 70 |
| Перелік посилань...………………………………………………………………………………... | 71 |
| Додаток А Програмний код виконаних досліджень …………………....………………………. | 73 |
| Додаток Б Слайди презентації…………………………………………………………………..... | 78 |
| Додаток В Тезиси до наукової конференції ……………………………………………............... | 84 |

Додаток Г Електронні матеріали………………………………………………………………….

**ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК**

СУБД - комплекс програм, що дозволяє створювати бази даних та маніпулювати ними (для надання, оновлення, видалення та вибору)

БД - сукупність даних, організованих відповідно до концепції, яка описує характеристику цих даних і взаємозв'язки між їх елементами; ця сукупність підтримує щонайменше одну з областей застосування.

NoSQL - механізм зберігання та видобування даних відмінний від підходу таблиць-відношень в реляційних базах даних.

BSON – двійковий формат представлення даних простих структур даних і асоціативних масивів

JSON - це текстовий формат обміну даними між комп'ютерами, що базується на тексті, може бути прочитаним людиною та надає змогу описувати об'єкти та інші структури даних

MS SQL - система управління реляційними базами даних, розроблена корпорацією Microsoft.

Redis - резидентна система управління базами даних класу NoSQL з відкритим вихідним кодом, що працює зі структурами даних типу «ключ – значення»

Cassandra - розподілена система управління базами даних, що належить до класу NoSQL-систем і розрахована створення високомасштабируемых і надійних сховищ великих масивів даних, представлених як хеша.

CRUD - акронім, що означає чотири базові функції, використовувані під час роботи з базами данних: створення (англ. create), читання (read), модифікація (update), видалення (delete).

**ВСТУП**

З розвитком інформаційних технологій почали з’являтися все нові можливості і розширюватися існуючі області їх застосування. Основний спосіб їх застосування полягає в створенні інформаційних систем, що працюють з певним обсягом даних і надають певні послуги. З ростом популярності також ростуть і розміри джерел даних, ростуть навантаження і, на жаль, падає продуктивність цих систем. Отже, це призводить до збільшення вимог до даних систем, серед яких можна відзначити: швидкість обробки даних, можливість лінійного масштабування, відмовостійкість обробки даних.

Завдання для виконання роботи є дослідження та порівняльний аналіз методів та моделей збереження даних ріхних видів СУБД.

Метою дослідження є порівняльний аналіз швидкісних та часових характеристик доступу до даних зі збільшенням навантаження, складності обслуговування, безпеки, фізичного зберігання та кількості спожитої па м’яті.

Задачами дослідження є огляд існуючих порівнянь, знаходження і обґрунтування необхідних критеріїв порівняння, виконання порівняльного аналізу за усіма знайденими критеріями та виконання висновку відповідно до зробленого аналізу відповідно найкращого способу зберігання даних у базі даних. А також вибір відповідного типу СУБД з найкращими результатами відповідно знайдених критеріїв.

Об'єктом дослідження є моделі збереження даних у реляційних та нереляційних базах даних та їх вплив на кількісні характеристики роботи з даними.

Методом вирішення задачі є розробка сервісу на за допомогою мови програмування C# з використання NoSQL баз даних, а також використанням реляційної бази даних MS SQL Server. Буде розроблено математичну модель, яка буде здатна проаналізувати отримані практичні розрахунки та допомогти прийняти рішення стосовно, відповісти на питання мети цього дослідження, яка модель зберігання даних у яких випадках має свої переваги та коли її застосування математично обґрунтовано.

Предметом дослідження є оцінка і порівняльний аналіз швидкісних характеристик виконання запитів, характеристик витрачання пам'яті і структурних характеристик (модифікація схеми бази даних) реляційних і нереляційних баз даних.

До певного моменту часу єдиним способом зберігання даних була реляційна СУБД. Але зі збільшенням обсягу даних виявлялися проблеми, з якими реляційна СУБД була не в змозі впоратися. У зв'язку з цим з'явилася необхідність пошуку розробки нових технологій зберігання даних, архітектура яких буде здатна адаптуватися до зростаючого обсягу даних і ефективно їх обробляти. Однією з таких технологій став перехід від реляційної моделі зберігання даних на постреляціонних модель і привів до появи NoSQL СУБД.

Наразі сховища даних такі як бази даних є невід'ємною частиною технічного забезпечення більшості підприємств і організацій. Основні класифікації баз даних можна охарактеризувати як реляційні та нереляційні. Вони мають дуже багато відмінних характеристик, які можуть допомогти у вирішенні проблеми у той чи інший час.

Як і для реляційних баз даних, так і для нереляційних баз даних характерний розвиток навіть після проектування та впровадження. Це пов'язано зі зміною характеру використання баз даних, кількістю користувачів, а так само набору наданих їм сервісів.

Для вирішення питань або проблем про необхідність вибору відповідного типу СУБД, необхідно проводити регулярний збір інформації про поточний стан змін та останніх модифікацій до відповідного типу СУБД, тобто виправлення якихось помилок або удосконалення якихось алгоритмів, наприклад удосконалення швидкості вибору даних з таблиць або взагалі проведення операцій роботи з цією таблицею.

Другий критерій, який необхідно брати до уваги, це цільове направлення конкретного типу СУБД для вирішення тих чи інших задач. Необхідно брати до уваги та звертати на те, що завжди треба керуватися поставленою для вирішення задачею. Та вже від неї відштовхуватися у тому чи іншому напрямку для вирішення питання вибору відповідного типу СУБД.

Вибір відповідного типу СУБД – це початкова архітектура Вашої майбутньої системи та має дуже великий внесок у подальший розвиток та масштабування, тому необхідно дуже зважено відноситися до цього етапу.

Теоретичною та методологічною основою дослідження в роботі слугували наукові дослідження та їх результати, опубліковані у науковій та професійній літературі.

У ході виконання роботи використано математичний метод дослідження. Створено математичну модель, тобто систему математичних співвідношень, які описують усі сторони, критерії, характеристики, яка буде використана для проведення аналізу. На етапі вибору типу математичної моделі за допомогою аналізу даних пошукового експерименту встановлено: лінійність або не лінійність, динамічність або статичність, стаціонарність або не стаціонарність, а також ступінь детермінованості досліджуваного об'єкта. Також за результатами створеної моделі проведено обчислювальний експеримент.

Раніше вже проводилися аналогічні дослідження стосовно порівняння характеристик СУБД. У більшості випадків ці дослідження порівнювали швидкісні характеристики виконання запитів для різних типів СУБД. Загалом у цих досліджень описувалися та проводилися експерименти з деяких базових запитів до СУБД та визначалися лише їх часові заміри. Наприклад ці дослідження виконували операції вставки 1000 елементів до бази даних різної наповненості та замірювали час виконання ціх операції. Тобто не давали точний результат продуктивності запитів при різній кількості даних, що оброблялися. Також унікальність цього дослідження у тому, що уділяється велике значення саме поєднанню СУБД з відповідною технологією, з якою вона має кращу сумісність, яка покращить результати у сукупністю за відповідними порівняльними характеристиками. Наприклад, реляційна СУБД MS SQL має показати кращі результати у поєднанні з технологією .Net, у той час, як СУБД MySQL має дати кращі результати у поєднанні з мовою програмування Python. Також буде удосконалено дослідження використовуючи основні оптимізаційні методи роботи з даними, наприклад використання індексів, максимальне оптимізація запитів (використання JOIN), максимальна оптимізація схеми бази даних, та ін.

Отримані результати можуть буди використанні при проектуванні інформаційної системи та полегшити вибір СУБД в залежності від розв'язуваної задачі та інших параметрів, які можуть поспливати та які розглядаються у поточному аналізі. Також отримані результати допоможуть прийняти рішення, чи варто змінювати старий реляційний підхід на більш новий та популярний з використанням NoSQL СУБД.

Наукова новизна роботи полягає в обґрунтуванні науково-методичних рекомендацій в області створення програмних продуктів, які взаємодіють з даними та базами даних, спрямованих на покращення продуктивності та ефективності цих продуктів на стадіях написання цих продуктів.

З основних результатів робіт можна виділити:

* наведена модель для порівняння баз даних різних СУБД;
* наведені фактори, які найбільше впливають на порівняльні характеристики;
* виявлені типи задач, поставлені цілі, для яких найкраще підходять дані СУБД, які можуть допомогти найкраще вирішити;
* проведений аналіз технічних, математичних характеристик та сформовані основні результати порівняння СУБД.

Результати були опубліковані у вигляді тезисів до наукової конференції «19-й Міжнародний молодіжний форум «Радіоелектроніка та молодь у XXI столітті»».

**1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ І ПОСТАНОВКА ЗАДАЧІ**

Враховуючи, що NoSQL СУБД доволі популярна та нова технологія збереження даних, вже було проведено досить багато досліджень та порівнянь NoSQL СУБД з традиційними реляційними СУБД [20]. Загалом ці дослідження були направлені на отримання часових характеристик при виконання звичайних CRUD операцій для різної кількості записів даних. Тобто вибиралася операція, наприклад додавання 1000-10000 записів до таблиці, та замірювався час виконання цієї операції [20].

Результати цих досліджень можна використовувати тільки на початкових стадіях проектування програмної системи та зробити вибір на користь тієї чи іншої СУБД. Але ці дослідження не повністю розкривають усі недоліки та переваги використання реляційної чи нереляційної СУБД у реальних програмних системах та не враховують потенційні проблеми, з якими можна зіштовхнутися під час використання тієї чи іншої СУБД. Наприклад, порівняння швидкісних характеристик на сервері з обмеженою оперативною пам’яттю у 512 МБ (враховуючи, що нереляційні СУБД поглинають дуже велику кількість оперативної пам’яті на сервері).

Також існує небагато якісних досліджень, які приділяють увагу не тільки самим СУБД, а й також технологіям, у поєднанням з деякими з них СУБД можуть дати дійсно більш кращі показники, наприклад ASP.Net + MS SQL або MySQL + PHP (або Python) [13].

Прихильниками концепції NoSQL підкреслюється , що вона не є повним запереченням мови SQL і реляційної моделі , проект виходить з того , що SQL - це важливий і дуже корисний інструмент , але при цьому він не може вважатися універсальним. Однією з проблем , яку вказують для класичних реляційних БД , є проблеми при роботі з даними дуже великого об'єму і у проектах з високим навантаженням . Основна мета підходу - розширити можливості БД там, де SQL недостатньо гнучкий , і не витісняти його там, де він справляється зі своїми завданнями [20].

Хороша горизонтальна масштабованість - автоматичний розподіл між декількома серверами.

Такі системи ми називаємо розподіленими базами даних (розподіл йде між комп'ютерами по локальній мережі). У них входять: Cassandra, HBase, Riak, Scalaris, Voldemort.

В якості одного з методологічних обґрунтувань підходу NoSQL використовується евристичний принцип, відомий як теорема Ca, який стверджує, що в розподіленої системі неможливо одночасно забезпечити узгодженість даних, доступність (англ. availability - коректність відгуку по будь-якому запиту) і стійкість до розщеплення розподіленої системи на ізольовані частини [21]. Таким чином, при необхідності досягнення високої доступності і стійкості до поділу планується не фокусуватися на засобах забезпечення узгодженості даних, забезпечуваних традиційними SQL - орієнтованими СУБД з транзакційними механізмами на принципах ACI [25].

У ході наукового дослідження у роботі необхідно провести порівняльний аналіз використання двох СУБД у поєднанні з відповідною мовою програмування, що робилося до цього у дуже малих дослідженнях та визначити найкраще поєднання СУБД та програмної системи для задачі зберігання ієрархічних даних у базі даних. Необхідно створити прикладне програмне забезпечення для визначення основних технічних та математичних характеристик, параметрів для проведення порівняння на відповідних технологіях з використання відповідних СУБД.

Також у ході дослідження планується зосередити особливу увагу методам та особливостям збереження даних у самих СУБД. Планується описати особливості основних внутрішніх елементів СУБД, їх недоліки та переваги та те, яким чином найкраще використати ці особливості внутрішньої будови.

Основну увагу треба зосередити на проведенні досліджень для поєднання СУБД із веб-додатками, тобто досліджувати основні характеристики необхідно для веб-додатків.

Планується вирішити наступні задачі при дослідженні поставленої цілі:

1. Дослідити моделі та методи збереження даних у реляційних та нереляційних БД;

2. Описати математичну модель порівняння та отримання кількісних характеристик кожної з СУБД;

3. Порівняти швидкісні характеристики запитів на стандарті SQL із запитами специфічних стандартів JSON (NoSQL);

4. Зробити аналіз на надійність оновлення даних, використовуючи транзакції в SQL і будь-які можливі засоби в NoSQL;

5. Порівняти обсяг споживаної пам'яті, а також відносну завантаження процесора при виконанні операцій звернення до бази даних;

6. Провести тестування навантаження при використанні відповідної БД на одному з серверів і порівняти отримані результати;

7. Дослідити надійність, безпека, структурованість при зміні / видаленні даних по зовнішньому ключу або іншому посилальному елементу В не реляційних БД;

8. Виділити загальні сильні сторони, загальні слабкі сторони, знайти та виявити можливі методи удосконалення та покращення продуктивності.

Результати вирішення задач проілюструвати необхідними графічними таблицями та графіками для більш наглядного та детального способу порівняння та визначення сильних і слабких сторін кожного з підходів збереження даних.

Метод дослідження - це спосіб отримання збору, обробки або аналізу даних. Основним орієнтиром для вибору методу дослідження можуть служити його завдання. Саме завдання, поставлені перед магістерською роботою, визначають способи їх дозволу і вибирають відповідні методи дослідження.

У ході наукового дослідження планується провести і вирішити пізнавальні завдання, які будуть з'являтися і будуть поставлені в ході магістерського дослідження при порівнянні і проведенні аналізу реляційних і не реляційних СУБД. Дані завдання будуть спрямовані на виявлення, точний опис і детальне вивчення різних фактів, явищ і процесів. Тому для виконуваного дослідження в магістерській роботі буде використовуватися емпіричний метод дослідження.

Відповідно до цього методу планується отримувати нові знання на основі дослідження за допомогою опису, спостереження і експерименту.

Також для проведення дослідження емпіричним методом, необхідно буде попередньо провести глибоке теоретичне обґрунтування щодо умов проведення дослідження, збору емпіричних даних для експерименту, обробки отриманих результатів і формулювання висновків, виходячи з проведеного дослідження.

Так як для дослідження було обрано емпіричний метод, то планується провести дослідження, що складається з наступних етапів в наступній послідовності: збір інформації, спостереження явищ, аналіз, розробка гіпотези для пояснення явища, розробка теорії, що пояснюють феномен, заснований на припущеннях.

Необхідно провести аналіз існуючих досліджень у сфері порівняння СУБД і визначити основні положення, проблеми, які недостатньо розкриті в поточних дослідженнях. Також необхідно усвідомити потребу в усуненні дефіциту і встановити, для чого і як в подальшому можуть бути використані дані дослідження в області порівняння характеристик СУБД. Також тут необхідно визначити наявність випробовуваних точок, тобто визначити місця, фактори для проведення експерименту, які будуть розглянуті і перевірені в ході експерименту.

Планується провести збір інформації, визначити основні складові математичної моделі, ґрунтуючись на допустимих технічних характеристиках, обраних реляційних і не реляційних СУБД. Потім на основі зібраної інформації та отриманих факторах впливу математичної моделі, планується проведення експериментів і спостереження основних явища, виходячи з поставлених завдань магістерського дослідження. Тобто планується проведення експериментів на порівняння і спостереження основних характеристик, таких як швидкість виконання запитів на вибірку СУБД, витрати оперативної пам'яті СУБД і т.д.

В ході проведення збору інформації також необхідно буде визначити об'єкт і предмет дослідження. Забігаючи наперед, можна уточнити, що для даного дослідження об'єктом дослідження є задача визначення критеріїв вибору конкретного типу БД (Не реляційного або реляційного) залежно від типу розв'язуваної задачі, а предметом дослідження є оцінка та порівняльний аналіз швидкісних характеристик виконання запитів, характеристик витрачання пам'яті і структурних характеристик (зміни схеми бази даних) реляційних і не реляційних баз даних.

Мета повинна буде коротким і гранично точно, в смисловому плані виражати те основне, що має намір зробити в магістерській дослідницької роботу. Потім мета буде конкретизована в задачах наукового дослідження, тобто кожна задача повинна чітким формулюванням розкривати якісь сторони досліджуваної теми. У ході аналізу і роботи над цим блоком будуть сформульовані основні завдання дослідження магістерської роботи. Перше завдання буде пов'язана з виявленням, уточненням, поглибленням, обґрунтуванням сутності, структури досліджуваного об'єкта. Друга буде пов'язана з аналізом реального стану предмета дослідження. Третє завдання буде пов'язана з перетворенням предмета дослідження, тобто виявлення шляхів і засобів підвищення ефективності вдосконалення методів, способів здійснення запитів до баз даних з отриманням найкращої продуктивності. Четверте завдання буде пов'язана з дослідно-експериментальною перевіркою ефективності, тобто буде перевірена поточна продуктивність реляційних і не реляційних СУБД, оцінені основні слабкі місця, які знижують продуктивність і уповільнюють час виконання та запропоновано основні методи оптимізації про поліпшення знайдених проблем. Наприклад, у випадку великої кількості часу на вибірку даних можливе використання індексів в базі даних, для операцій вставки і оновлення даних, можливо заздалегідь видалення цих індексів і їх подальше відновлення після виконання операцій.

Для успішного проведення експерименту необхідно буде мати в наявності 2 бази даних, наповнені готовими даними достатнього обсягу, щоб мати можливість зробити основні часові виміри (щоб була достатня швидкість виконання). Також необхідно буде вирішити питання щодо платформи проведення експерименту (операційної системи). Для оцінки результатів проведення експериментів буде створено програмне забезпечення, що перевіряє і надає основні етапи, і результати дослідження.

У ході обробки буде використана математична модель обробки даних, тобто буде визначається характер розподілу по всіх використовуваних параметрах дослідження.

Наприкінці будуть сформульовані висновки, що містять підсумки дослідження, тобто містять припущення про те, яка СУБД є більш придатною для вирішення поставленого завдання, які їхні сильні і слабкі сторони були виявлені в ході дослідження і які поради можна винести, виходячи з даного дослідження.

**2 РЕЛЯЦІЙНІ ТА НЕРЕЛЯЦІЙНІ МОДЕЛІ ЗБЕРЕЖЕННЯ ДАНИХ**

2.1 Опис реляційної та не реляційної моделі

СУБД ґрунтуються на моделі бази даних - це спеціальні структури призначені для роботи з даними. Всі СУБД сильно відрізняються в тому, яким чином вони зберігають і обробляють свої дані [4].

Хоча існують багато рішень для роботи з БД, популярними і затребуваними стають лише деякі з них. Найбільш часто застосовувана на сьогоднішній день - реляційна система управління базами даних.

Кожна система підтримує різні моделі і структури баз даних. Ця модель і визначає, як створювана СУБД буде оперувати даними. Існує досить небагато моделей БД, які надають способи чіткого структурування даних, найпопулярніша з таких моделей - реляційна модель [5].

Реляційна модель і реляційні БД можуть бути дуже потужним інструментом, але тільки якщо програміст знає як з ними поводитися [5]. Нещодавно, стали набирати популярність NoSQL системи з обіцянкою позбутися старих проблем БД і додати новий функціонал. Виключаючи жорстку структуру даних, при цьому зберігши реляційний стиль, ці СУБД пропонують більш вільний спосіб роботи з ними і набагато більші можливості для їх налаштування. Хоча не обходиться і без виникнення нових проблем.

Реляційна модель представлена в 1970 році запропонувала математичний спосіб структурування, збереження і використання даних. По суті він розширив плоску і мережеву моделі, об'єднавши їх в реляційну. Основна перевага якої було об'єднання даних в групи, саме реляційна модель дозволила зберігати дані в структурованому табличному вигляді (ПІБ, адреса) [23].

Завдяки десятиліттям розробки, СУБД досягли досить високого рівня в продуктивності та відмовостійкості. Досвідом розробників і мережевих адміністраторів було доведено, що всі ці інструменти відмінно справляються зі своїми функціями в додатках будь-якої складності, не втрачають даних навіть при некоректних завершеннях роботи [23].

Незважаючи на великі обмеження у формуванні та управлінні даними, реляційні бази даних зберігають широкі можливості по налаштуванню і пропонують досить великий функціонал [20].

NoSQL намагається прибрати всі обмеження реляційної моделі (недостатня продуктивність, трудомістке горизонтальне масштабування, недостатня продуктивність в кластері) і полегшує засоби зберігання і доступу до даних. Такі БД використовують неструктурований підхід (створення структури на льоту), тим самим знімаючи обмеження жорстких зв'язків і пропонуючи різні типи доступу до специфічних даними [20].

Реляційні СУБД беруть свою назву від моделі БД з якою працюють. На даний момент і, напевно, в найближчому майбутньому ці СУБД будуть найбільш популярним вибором для зберігання даних.

Реляційні СУБД використовують строго описані структури даних - схеми. Схема бази даних включає в себе опис змісту, структури і обмежень цілісності, тобто вона визначає таблиці, поля в кожній таблиці, а також відносини між полями і таблицями.

Ось деякі з найбільш популярних систем:

* SQLite - досить потужна вбудована СУБД;
* MySQL - найбільш популярна СУБД;
* PostgreSQL - сама професійна вільно поширювана СУБД, повністю відповідає стандартам SQL.

Концепції реляційної моделі вперше були сформульовані в роботах американського вченого Е. Ф. Кодда. Звідки походить її друга назва - модель Кодда [5].

У реляційній моделі об'єкти і взаємозв'язки між ними представляються за допомогою таблиць (рис. 2.1). Для її формального визначення використовується фундаментальне поняття відносини. Власне кажучи, термін "реляційна" походить від англійського relation - відношення. Якщо задані довільні кінцеві множини D1, D2, ..., Dn, то декартовим твором цих множин D1? D2? ...? Dn називають безліч всілякі наборів виду (d1, d2 ..., dn), де

d1 D1, d2 D2, ..., dn Dn. Відношенням R визначеним на множинах D1, D2, ..., Dn ,, називається підмножина декартова твори Dl x D2x ... х Dn. При цьому безлічі D1? D2? ...? Dn називаються доменами відносини, а елементи декартова твори - кортежами відносини. Число я визначає ступінь відносини, а кількість кортежів - його потужність. Поряд з поняттями домену та кортежу при роботі з реляційними таблицями використовуються альтернативні ним поняття поля і записи [5].

У реляційній базі даних кожна таблиця повинна мати первинний ключ (ключовий елемент) - поле або комбінацію полів, які єдиним чином ідентифікують кожен рядок в таблиці [25].

Важливою перевагою реляційної моделі є те, що в її рамках дії над даними можуть бути зведені до операцій реляційної алгебри, які виконуються над відносинами. Це такі операції, як об'єднання, перетин, віднімання, декартовій твір, вибірка, проекція, з'єднання, ділення.

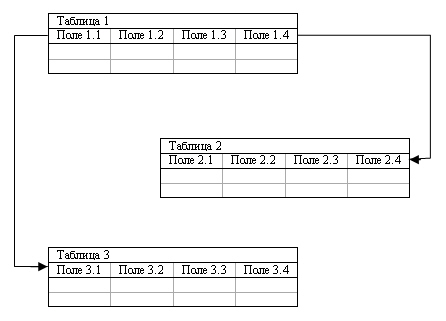


Рисунок 2.1 – Схема реляційної моделі даних

Найважливішою проблемою, розв'язуваної при проектуванні баз даних, є створення такої їх структури, яка б забезпечувала мінімальне дублювання інформації і спрощувала процедури обробки та оновлення даних. Код-будинок був запропонований деякий набір формальних вимог універсального характеру до організації даних, які дозволяють ефективно вирішувати перераховані завдання.

Ці вимоги до стану таблиць даних отримали назву нормальних форм. Спочатку були сформульовані три нормальні форми. Надалі з'явилася нормальна форма Бойса-Кодда і нормальні форми більш високих порядків. Однак вони не набули широкого поширення на практиці [23].

- Кажуть, що ставлення знаходиться в першій нормальній формі, якщо всі його атрибути є простими.

- Кажуть, що ставлення знаходиться в другій нормальній формі, якщо воно задовольняє вимогам першої нормальної форми і кожен не ключовий атрибут функціонально повно залежить від ключа (однозначно визначається ним).

- Кажуть, що ставлення знаходиться в третій нормальній формі, якщо воно задовольняє вимогам другої нормальної форми і при цьому будь-який не ключовий атрибут залежить від ключа нетранзитивно. Зауважимо, що транзитивної називається така залежність, при якій будь-якої не ключовий атрибут залежить від іншого не ключового атрибуту, а той, у свою чергу, вже залежить від ключа.

Принциповим моментом є те, що для приведення таблиць до стану, що задовольняє вимогам нормальних форм, або, як ще кажуть, для нормалізації даних над ними, повинні бути здійснені перераховані вище операції реляційної алгебри [4].

Основною перевагою реляційної моделі є її простота. Саме завдяки їй вона покладена в основу переважної більшості реально працюючих СУБД [4].

У розробленій Коддом реляційної моделі були визначені як вимоги до організації таблиць, що містять дані, так і мова, що дозволяє працювати з ними. Згодом ця мова отримав назву SQL (Structured Query Language - структурована мова запитів) [4]. SQL був вперше реалізований фірмою I на початку 70-х років двадцятого століття під назвою Structures English Query Language (SEQUEL). Він був зорієнтований на управління прототипом реляційної бази даних IBM-System R. Надалі SQL став стандартом de facto мови роботи з реляційними базами даних. Цей його статус був вперше зафіксований в 1986 році Американським національним інститутом стандартів (ANSI).

Іншими досить відомими стандартами SQL стали стандарти ANSI SQL-92 ISO SQL-92, X / Open. У складі SQL можуть бути виділені наступні групи інструкцій:

- Мова опису даних - DDL (Data Definition Language);

- Мова маніпулювання даними - DML (Data Manipulation Language);

- Мова управління транзакціями.

Інструкції DDL призначені для створення, зміни та видалення об'єктів бази даних. Їх опис наведено в табл. 2.1.

Таблиця 2.1. Інструкції мови визначення даних (DDL)

|  |  |
| --- | --- |
| Інструкція | Призначення |
| CREATE | Створення нових об'єктів (таблиць, полів, індексів і т. Д.) |
| DROP | Видалення об'єктів |
| Продовження таблиці 2.1 | |
| ALTER | Зміна об'єктів |

Інструкції DML (табл. 2.2) дозволяють вибирати дані з таблиць, а також додавати, видаляти і змінювати їх.

Таблиця 2.2. Інструкції мови маніпулювання даними (DML)

|  |  |
| --- | --- |
| Інструкція | Призначення |
| SELECT | Виконання запиту до бази даних з метою відбору записів, що задовольняють заданим критеріям |
| INSERT | Додавання записів у таблиці бази даних |
| UPDATE | Зміна значень окремих записів і полів |
| DELETE | Видалення записів з бази даних |

SELECT - команда на вибірку записів з бази даних - є найбільш часто використовуваною SQL-інструкцією. Сфера даних, якими вона маніпулює, визначається за допомогою спеціальних пропозицій. Перелік основних пропозицій мови SQL наведено в табл. 2.3.

Таблиця 2.3. Основні пропозиції мови SQL

|  |  |
| --- | --- |
| Інструкція | Призначення |
| FROM | Вказує ім'я таблиці, з якої повинні бути відібрані дані |
| WHERE | Специфікує умови, яким повинні задовольняти вибирані дані |
| GROUP BY | Визначає, що вибрані записи повинні бути згруповані |
| HAVING | Задає умова, якому повинна задовольняти кожна група відібраних записів |
| ORDER BY | Специфікує порядок сортування записів |

Третьою складовою частиною SQL є мова управління транзакціями. Транзакція - це логічно завершена одиниця роботи, що містить одну або більше елементарних операцій обробки даних. Всі дії, що становлять транзакцію, повинні або виконатися повністю, або повністю не виконатися. Інструкції мови управління транзакціями наведено в табл. 2.4.

Таблиця 2.4. Інструкції мови управління транзакціями

|  |  |
| --- | --- |
| Інструкція | Призначення |
| COMMIT | Фіксація в базі даних всіх змін, зроблених поточної транзакцією |
| SAVEPOINT | Установка точки збереження (початку транзакції) |
| ROLLBACK | Відкат змін, зроблених з моменту початку транзакції |

У більшості СУБД елементарні команди, складові тіло транзакції, виконуються над деякою буферної копією даних, і тільки якщо їх вдається успішно довести до кінця, відбувається остаточне оновлення основної бази. Транзакція починається від точки збереження, що задається інструкцією SAVEPOINT, і може бути завершена по команді COMMIT або перервана по команді ROLLBACK (відкат). Також в сучасних системах управління даними передбачені кошти автоматичного відкоту транзакцій при виникненні системних збоїв. Таким чином, механізм управління транзакціями є найважливішим інструментом підтримки цілісності даних [5].

NoSQL бази даних не працюють з реляційними моделями. Існує багато різних рішень, кожне з яких працює трохи по-своєму і служить специфічної мети. Ці без схемні рішення знімають обмеження з формування сутностей і допускають зберігання даних у вигляді ключ-значення [20].

На відміну від реляційних баз даних, можна групувати колекції даних з іншими NoSQL базами даних, наприклад MongoDB. Такі СУБД зберігають дані як одне ціле в базі. Такі дані можуть являти собою одиночний об'єкт як JSON і разом з тим коректно відповідати на запити до полів [20].

NoSQL бази даних не використовують загальний формат запиту, такий як SQL в реляційних базах даних. Кожне NoSQL рішення використовує власну систему запитів [20].

Для уявлення загальної картини давайте порівняємо ці два типи СУБД:

* Структури даних та їх типи - реляційні БД використовують строгі схеми даних, NoSQL БД допускають будь-який тип даних;
* Запити - незалежно від типу ліцензії, реляційні бази даних в тій чи іншій мірі відповідають стандартам SQL, тому дані з них можна отримувати за допомогою мови SQL. NoSQL БД використовують специфічні способи запитів до даних;
* Масштабованість - обидва ці типи СУБД досить легко піддаються вертикальному масштабуванню (тобто збільшення системних ресурсів). Тим не менш, так як NoSQL це більш сучасний продукт, саме такі СУБД пропонують більш прості способи горизонтального масштабування (тобто створення кластера з декількох машин);
* Надійність - коли справа доходить до збереження даних та гарантії виконання транзакцій SQL БД по колишньому займають лідируючі позиції;
* Підтримка - реляційні СУБД мають не малу історію за плечима. Вони дуже популярні і пропонують як платні, так і безкоштовні рішення. При виникненні проблем, все ж набагато простіше знайти відповідь, якщо справа стосується реляційних систем, ніж NoSQL, особливо якщо рішення досить складне за своєю природою (наприклад MongoDB);
* Зберігання та доступ до складних структурам даних - спочатку реляційні системи припускали роботу зі складними структурами, саме тому вони перевершують інші рішення по продуктивності.

За задумом NoSQL бази даних і СУБД не мають на увазі внутрішніх зв’язків. Вони не ґрунтуються на одній моделі, а кожна база даних в залежності від цілей використовує різні моделі.

Існує досить багато різноманітних моделей та функціональних систем для NoSQL баз даних [15]:

* Сховище ключ-значення - Redis, MemcacheD і т.д. (зазвичай зберігають дані в пам'яті);
* Розподілене сховище (Column-oriented) - Cassandra, HBase і т.д (призначені для дуже великих обсягів даних);
* документо-орієнтовані - MongoDB, Couchbase і т.д. (призначені для зберігання ієрархічних структур даних - документів);
* БД на основі графів - OrientDB, Neo4J і т.д.

**Сховище ключ-значення** є основним рішенням сімейства NoSQL. Цей тип БД працює з даними типу ключ-значення, наприклад як словник. Тут немає місця ні структурі, ні зв’язкам. Після підключення до сервера (наприклад Redis) додаток може задати ключ і його значення, а в наслідку отримувати ці дані за запитом [20].

Такі СУБД зазвичай використовуються для швидкого збереження базових даних, а іноді не таких уже й базових, якщо підрахувати витрати процесора і пам'яті. Вони, зазвичай, дуже швидкі, працездатні або легко масштабуються (добре використовувати такі БД для зберігання сесій, кеша, лічильників відвідувань або переглядів і т.д.).

Такі БД дуже продуктивний, прості в обігу і легко масштабуються.

Деякі популярні сховища:

* Redis - БД в оперативній пам'яті з додатковою відмово стійкістю;
* Riak - Розподілене, реплікаційне сховище;
* Memcached / MemcacheDB - розподілена БД в оперативній пам'яті.

Часто зустрічаються випадки застосування БД сховищ ключ-значення:

* Кешування - швидке і часте збереження даних для майбутнього використання;
* Черга - деякі БД типу ключ-значення підтримують списки, набори і черги;
* Розподіл інформації / завдань - використовується для реалізації патерна Pub / Sub;
* Живе оновлення інформації - додатки використовують стану.

**Розподілене сховище** по суті це наступний крок після СУБД типу ключ-значення. Незважаючи на досить складну для розуміння сутність, ці бази даних відмінно працюють просто створюючи колекції з одного або декількох пар ключ-значення, які в сумі відповідають запису [20].

На відміну від звичних таблиць в реляційних моделях, ці СУБД не вимагає попереднього опису структури даних. Кожен запис складається з одного або декількох стовпців містять дані, а кожен стовпець різних записів може зберігати різні типи даних.

В цілому, розподілене сховище ні що інше як двовимірний масив, де кожен ключ (запис) містить одну або кілька пар ключ-значення прив'язаних до нього. Така система дозволяє зберігати і використовувати великі обсяги неструктурованих даних. (один запис з великим кількість додаткової інформації). Такі бази даних зазвичай використовуються, коли недостатньо простих пар ключ-значення, а зберігати необхідно великий обсяг записів з різною інформацією. Такі СУБД при відповідному використанні можуть бути дуже продуктивними [20].

Такі системи баз даних дуже ефективні і можуть бути використані для зберігання важливої інформації великих обсягів. Може вони де щось не дуже гнучкі в плані даних, зате вони функціональні і продуктивні.

Основні представники даного типу:

* Cassandra - структура даних заснована на BigTable і DynamoDB;
* HBase - сховище для Apache Hadoop засноване на принципах BigTable.

Основні області застосування:

- Зберігання неструктурованих, не руйнується даних - якщо вам необхідно зберігати великі обсяги даних протягом довгого часу, то такі БД дуже добре впораються із завданням;

- Масштабування - за задумом такі бази даних легко масштабуються. Вони легко справляються з будь-яким обсягом даних.

**Документо-орієнтовані сховища** дуже швидко захопили свій ринок. Вони працюють так само як і попередні системи, але вони допускають набагато більшу вкладеність і складність структури даних. (наприклад, документ вкладений в документ, вкладений у документ). Документи знімають обмеження вкладеності першого та другого рівнів типу ключ-значення в розподілених сховищах. В цілому, можна описати як завгодно складну структуру даних як документ і зберегти в такій БД [20].

Незважаючи на досить великий функціонал і здатність доступу до даних по одному ключу, такі СУБД мають ряд своїх проблем. Наприклад, при доступі до одного документа ви повністю отримуєте його у відповідь на запит, навіть якщо вам необхідно якесь одне поле, що не може не позначитися на продуктивності [15].

Документ-орієнтовані сховища відмінно зберігають незв'язану інформацію великих обсягів, навіть якщо вона дуже різниться від сутності до сутності.

Популярні СУБД:

* Couchbase - засноване на JSON, сумісний з Memcached сховище;
* CouchDB - передове документо-орієнтоване сховище;
* MongoDB - дуже популярне і функціональне сховище.

Часто зустрічаються сфери застосування:

* Вкладена інформація - документо-орієнтовані сховища відмінно працюють з глибоко вкладеної, складною інформацією;
* Підтримка JavaScript - одна з відмінних особливостей документо-орієнтованих сховищ це те, як вони працюють з іншими додатками: підтримка JSON.

**Бази даних на основі графів** зберігають дані в зовсім іншому вигляді. Вони використовують деревовидні структури з вузлами і зв'язками їх з'єднують. Так само як і в математиці, деякі операції набагато зручніше виконувати з такими даними завдяки зв'язкам між ними та їх угрупованню (наприклад, зв'язки між людьми) [15].

Такі бази даних часто використовуються в додатках, де потрібно мати чітко встановлені зв'язку. Наприклад, коли ви реєструєтеся в будь якої соціальної мережі, зберігати зв'язку між вами і вашими друзями зберігати набагато простіше при використанні баз даних на основі графів.

Популярні СУБД:

* OrientDB - дуже швидке документо-орієнтоване сховище гібрид типу граф написане на Java. Включає в себе різні режими роботи;
* Neo4J - безсистемне, дуже потужне і популярне сховище написане на Java.

Сфери застосування:

* робота зі складною пов'язаної інформацією - як було сказано у вступі, сховища типу граф відмінно справляються зі складно пов'язаної інформацією. Наприклад зберігання зв'язків між двома сутностями і цілого ряду різнорівневих зв'язків між сутностями не пов'язаних з першими безпосередньо;
* Моделювання та підтримка класифікацій - такі БД досягли успіху скрізь де є зв'язку. Моделювання даних і класифікація різної інформації по зв'язках можна з легкістю уявити використовуючи ці БД.

Реляційна база даних може розглядатися як набір таблиць; з нею можна звертатися відповідно до реляційної моделі даних. Кожна база даних включає в себе:

* набір таблиць системного каталогу, що визначає логічну й фізичну структуру даних;
* файл конфігурації, що містить значення параметрів, які застосовуються для цієї бази даних;
* журнал відновлення з виконуються і архівувати транзакціями.

У реляційній моделі база складається з таблиць, які складаються з рядків і колонок. У кожної колонки є свій тип даних (рядок, число, логічне значення, дата, текст, бінарний блоб). Всі рядки однотипні [15].

Зазвичай кожен вид об'єктів зберігається в окремій таблиці (наприклад, таблиця користувачів або таблиця проектів). Зазвичай у кожного об'єкта є унікальний ідентифікатор. Ідентифікатор може бути як умовним, тобто просто числом, так і випливають з предметної області, наприклад номер паспорта людини або ISBN для книг. Зазвичай користуються умовними ідентифікаторами.

Об'єкти (таблиці) можуть бути пов'язані один з одним за допомогою ідентифікатора. Наприклад, якщо у нас є таблиця відділів і таблиця співробітників, то в таблиці відділів є ідентифікатор відділу, а в таблиці співробітників є ідентифікатор співробітника і ідентифікатор відділу, до якого він належить. В теорії реляційних баз даних цей випадок називається "один до багатьох" (одному відділу належить багато співробітників) [20].

Можливий також випадок "багато до багатьох". Наприклад, є таблиця проектів і таблиця розробників. Над одним проектом можуть працювати багато розробників, і один розробник може працювати над декількома проектами. У цьому випадку зазвичай створюється третя таблиця - таблиця зв'язків з двома полями: ідентифікатором проекту і ідентифікатором розробника. Кожна зв'язок між розробником і проектом виражається у вигляді рядка в таблиці зв'язків. Якщо розробник поки ще не призначений ні на один проект, то в таблиці зв'язків просто не буде жодного запису про нього [20].

Сервери реляційних БД забезпечують стандартні оператори доступу до даних в таблицях, такі як операції SELECT, INSERT, UPDATE і DELETE. Різні сервери надають також деякі додаткові оператори. Отримувати дані з таблиць можна по безлічі різних критеріїв. Є "ядро" стандарту SQL, який підтримується практично всіма серверами, і завжди є ті чи інші розширення стандарту, які можна використовувати при роботі з конкретним сервером БД.

"Жорсткість" реляційної моделі даних дозволяє реалізувати різні оптимізації доступу до даних. Найбільш очевидний приклад - це створення індексів по полях таблиці для швидкого доступу. Наприклад, на таблиці співробітників можна створити індекс на поле "ідентифікатор відділу", і тоді операція "отримати список співробітників того чи іншого відділу" працюватиме швидше. Індекс - це просто матеріалізована структура даних, така як B-дерево або хеш. Важливо розуміти, як влаштована ця структура даних, щоб можна було робити висновки про те, як вона буде працювати в тому чи іншому випадку.

Важливу роль у проектуванні реляційної БД грає нормалізація і денормалізація моделі даних. Нормальна форма БД - це така, де інформація не повторюється. Для швидкості та ефективності іноді базу даних денормалізують, і тоді в ній з'являється дублююча інформація [20].

Наприклад, у нас є таблиця клієнтів і таблиця продажів. Деякі клієнти вважаються "важливими", тому що вони купили на суму більше N. Ми могли б кожен раз для кожного клієнта витягувати список його продажів, підсумовувати вартість і порівнювати її з N. У той же час для швидкості ми можемо додати в таблицю клієнтів поле - прапорець "важливий" і постійно підтримувати його в цілісному стані - наприклад, при закладі нового продажу перевіряти, чи стала загальна сума більше N і якщо так, то виставляти цей прапорець в "TRUE". При помилках програмування такі поля можуть розсинхронізуватися і тоді базу даних доводиться "лагодити".

Вдала денормалізація може сильно збільшити продуктивність. Однак, денормалізація - це не панацея, вона може привести і до негативних наслідків.

Як ефективно працювати в реляційної БД з такими структурами даних, як ієрархічне дерево або граф? За багато років накопичені величезний досвід в цій області. Наприклад, ієрархічні дерева для швидкості можна зберігати з допомогою materialized path. Дані в стилі key-value можна зберігати як у вигляді очевидною трьох-колонкової таблиці "id\_об'екта-ключ-значення", так і (іноді) у вигляді "широкої" таблиці.

Для деяких структур даних, починаючи з певного розміру, практично неможливо ефективно укластися в реляційну БД, і доводиться використовувати спеціалізовані рішення. Наприклад, граф дружніх зв'язків між мільярдом людей практично неможливо обробляти за допомогою стандартних графових алгоритмів в рамках реляційної моделі навіть на сучасному обладнанні.

Одне із значень терміну "NoSQL" - це відхід від реляційної моделі на користь більш специфічних (або більше узагальнених) моделей даних. Наприклад, традиційно успішними NoSQL-системами є системи зберігання пар "ключ-значення", такі як Redis або Memcache. Їхня модель даних гранично проста - це по суті асоціативний масив, де ключі мають строковий тип, а значення можуть містити будь-які дані. Як і будь асоціативний масив, такі системи підтримують обмежений набір операцій з даними - прочитати значення по ключу, встановити значення ключа, видалити ключ і пов'язане з ним значення. Операція "отримати список ключів" може не підтримуватися в таких системах [14].

Інший приклад успішних NoSQL-систем - це документні сховища. Об'єкти в таких сховищах зазвичай є асоціативними масивами вільної структури, тобто в одній і тій же "таблиці" можуть зберігатися різні за суттю об'єкти. Приклади систем такого класу - MongoDB і Cassandra. Залежно від того, які реально дані зберігаються в конкретній базі, її продуктивність може сильно варіюватися. Наприклад, якщо оптимізувати таку "таблицю", зберігаючи в ній однотипні об'єкти, то ми можемо побудувати індекс по одному або декількох полях [15].

Третій приклад спеціалізованих NoSQL-систем - це графові бази даних. Вони спеціальним чином заточені під обробку конкретної структури даних, причому зазвичай для роботи з великим обсягом даних (бо на невеликих обсягах може чудово впоратися стандартна реляційна реалізація).

Дуже важливим прикладом NoSQL-систем є звичайні файлові системи, такі як Ext4 або NTFS. Вони призначені для зберігання об'єктів у вигляді ієрархічної структури з вмістом вільного формату. Самі бази даних, реляційні та NoSQL, зазвичай використовують для зберігання свого вмісту саме файлові системи, і іноді взаємодія між цими двома підсистемами стає важливим в тому чи іншому випадку.

Ще один важливий випадок - системи повнотекстового пошуку, такі як Elastic Search або Google Search Engine.

Принципова проблема проектування системи, що використовує бази даних, полягає в тому, що на порівняно невеликих обсягах даних працює практично будь-яка система, а при порівняно великих обсягах даних - поступово перестає працювати практично будь-яка система. Це означає, що в процесі розвитку системи і збільшення обсягу даних доводиться заново продумувати роботу з даними, міняти модель зберігання даних або навіть замінювати сервер БД на інший.

Традиційно вважається, що збільшення обсягу даних на кожен наступний порядок вимагає перепроектування бази даних. Іноді з цим намагаються боротися, проектуючи базу відразу на два-три порядки вперед, проте це не завжди можливо в повній мірі. Питання роботи з збільшуються даними - невирішена в загальному випадку інженерне завдання.

Інший стандартної проблемою є раптово з'являється необхідність застосовувати до вже існуючих даними нові алгоритми, зазвичай з високими вимогами по швидкості. Наприклад, в деякій компанії зберігається вся інформація про продажі товарах, придатна для бухгалтерії та щомісячних звітів. Однак виклики часу вимагають почати щодня і щогодини аналізувати інформацію про історію продажів і приймати бізнес-рішення на основі цього аналізу - в які магазини направляти товари, які рекламні кампанії стартувати, що ще пропонувати людям, які купують ті чи інші товари. Такі алгоритми можуть зажадати принципово змінити спосіб зберігання даних, при цьому зберігаючи сумісність з існуючою системою і з існуючими даними. Питання роботи в таких умовах - невирішена інженерне завдання.

Будь-яке обладнання рано чи пізно відмовить: диски, пам'ять, процесор, електричне живлення і т. Д. У цьому розділі ми розглянемо випадок однієї фізичної машини, на якій крутиться сервер. Нехай у цій фізичної машини раптово пропадає харчування. Після відновлення живлення вона знову завантажується і запускає сервер БД. Що станеться з даними?

Кожна система БД, реляційна і NoSQL, має свою стратегію обробці таких відмов.

Взагалі кажучи, можлива "нульова стратегія", коли всі дані просто губляться і база даних стає порожньою. Прикладом вкрай успішною NoSQL-системи з такою стратегією є Memcache.

Реляційні системи БД традиційно підтримують ту чи іншу стратегію, яка забезпечує набір гарантій, який називається ACID: atomicity, consistency, isolation, durability (атомарность, узгодженість, ізоляція, надійність). Ці терміни відносяться до обробки транзакцій.

Транзакція - це набір операцій, які розглядаються як єдине ціле. Класичний приклад транзакції - це пересилання грошей між двома банківськими рахунками. Для цього ми повинні зменшити суму на одному рахунку і одночасно з цим збільшити суму на іншому рахунку.

Atomicity (атомарность) - це гарантія того, що при будь-якому поведінці обладнання або буде виконано обидві цих операції, або не виконано жодної. Тобто навіть якщо ми "знімемо гроші з одного рахунку", і в цю мікросекунду відбудеться стрибок напруги - після перезавантаження бази та введення її в робочий режим ми знову побачимо колишню суму на вихідному рахунку.

Consistency (узгодженість) - це найменш чітко певна гарантія. Крім того, цей термін ще й використовується у визначенні CAP-теореми (про яку див. Нижче), і там він означає щось інше (але близьке). Найбільш загально можна сказати, що узгодженість гарантує деякий "розумне" поведінка бази даних, таке що програміст не отримає особливих сюрпризів при роботі з базою даних, а також при відмовах устаткування.

Isolation (ізоляція) означає, що під час виконання транзакції інші паралельно виконуються операції "не бачать" проміжний стан. Наприклад, ми порахували загальну суму на рахунках. Тепер, якщо ми почнемо виконувати пересилання грошей, "знімемо гроші з одного рахунку" і в цю мікросекунду інший процес знову спробує порахувати "загальну суму на рахунках", то ми отримаємо колишню суму, і не менше. Durability (надійність) означає, що після успішного завершення транзакції її результати вже не будуть втрачені ні за яких умов. Наприклад, ми виконаємо пересилання грошей, закриємо транзакцію і отримаємо від сервера повідомлення про успішне завершення транзакції. Через мікросекунду відбудеться стрибок напруги. Надійність гарантує, що коли машина знову завантажиться і ввійде в робочий режим - інформація про пересилання грошей збережеться в базі даних.

Традиційно бази даних, що підтримують ACID, дозволяють і до деякої міри порушувати його, за допомогою т. Зв. "Transaction isolation level" (рівнів ізоляції транзакцій). Наприклад, на рівні "uncommitted read" паралельні транзакції можуть "побачити" проміжні стани інших транзакцій.

Взагалі, ослаблення гарантій часто дозволяє збільшувати ефективність ціною особливих вимог до інтерпретації результатів (так, вони можуть бути недостатньо точними або просто некоректними). Для деяких випадків це може бути виправдане: наприклад, якщо ми хочемо показати на сайті загальна кількість зареєстрованих користувачів, то в загальному нам нецікаво якнайточніше значення - досить сказати "близько сотні" або просто показати "якесь" число, тому що ніхто ніколи не зможе підтвердити або спростувати кожне конкретне значення.

Багато NoSQL-системи просто відмовляються від підтримки ACID, і замість цього оголошують якийсь свій унікальний набір гарантій, який може бути де завгодно в діапазоні від "нульового" до більш-менш близького до "повного ACID". Наприклад, деякі версії деяких систем просто можуть при відмові машини залишити базу даних в пошкодженому стані, так що буде потрібно її ручне або напівавтоматичне відновлення після перезавантаження, при цьому не гарантується, що всі записані дані при цьому будуть збережені.

Гарантії, ослаблені на рівні окремої машини, можна "відновити" або навіть побудувати на їх основі істотно більш надійну систему, якщо об'єднати фізичні машини в мережу і зажадати спеціального режиму роботи з ними. Про це докладніше див. Нижче.

Зазвичай у бази даних багато клієнтів, які паралельно роблять операції як читання, так і записи. База даних повинна в цьому випадку виконувати гарантії, які в ній закладені. Наприклад, реляційні БД зазвичай забезпечують ізоляцію транзакцій (див. Вище).

Підтримка паралельного доступу до баз даних часто вимагає істотних зусиль від розробника сервера, який повинен забезпечити швидкість і надійність такого доступу. Є багато різних алгоритмів і структур даних, що лежать в основі паралельного доступу.

Наприклад, щоб додати запис в таблицю, нам треба виділити нову сторінку в таблиці, а також оновити індекс. Якщо паралельно інший клієнт додає інший запис, то йому треба виділити ще одну сторінку (або скористатися тією ж самою?), І знову оновити індекс (або може бути поєднати дві операції оновлення індексу?). Що якщо перший клієнт почав транзакцію, анонсував додавання запису, почекав дві секунди, і відкотив транзакцію? Що якщо один клієнт збільшив значення поля на одиницю, а другий зменшив на одиницю? Що якщо в будь-яку мікросекунду може відбутися стрибок напруги, і система після перезавантаження повинна повернутися до "коректному" станом, незважаючи на всі численні комбінації проміжних умов і станів?

Дотримання гарантій в умовах паралельного доступу при збереженні продуктивності - величезна і складна інженерна задача. Всі сервери БД вирішують її за допомогою більш-менш стандартних підходів, проте конкретна реалізація цих підходів, і тонкощі, з ними пов'язані - різні в кожному сервері БД.

Традиційно вважається, що збільшення на порядок кількості одночасних клієнтів бази даних вимагає перегляду її архітектури. У загальному вигляді це питання - невирішена інженерне завдання.

Всі сервери баз даних забезпечують безліч адміністративних функцій, пов'язаних з життям сервера на окремій машині. Серед таких функцій - резервне копіювання; відновлення з резервної копії; оптимізація місця, займаного таблицями; розподіл файлів з даними з різних дискам і файловим системам; мережевий доступ до сервера БД (див. також відповідну главу книги) і ефективність такого доступу.

Також деякі сервера вміють ефективно використовувати спеціальні функції операційної системи (найчастіше, у свою чергу, розроблені спеціально для серверів БД). Типовий приклад - підтримка асинхронного введення-виведення.

Є також адміністративні функції, пов'язані з об'єднанням фізичних машин в мережу. Наприклад, це настройка топології реплікації, а також управління машинами в складі кластерів. Докладніше про це див. Нижче.

Всі ці, і багато інших функцій, по-своєму реалізовані в різних серверах баз даних. Адміністративні функції, їх пропрацьованість і зручність, є важливим критерієм вибору сервера БД, відповідного для конкретного завдання.

Сучасні сервери баз даних забезпечують безліч тонких налаштувань продуктивності. Порівняння швидкості роботи різних БД в спеціальних умовах - захоплююче і не завжди осмислене заняття.

Важливо розуміти, що будь настроєний під деякі умови сервер завжди можна "поставити на коліна", змінивши патерни доступу до даних, збільшивши кількість клієнтів або збільшивши кількість збережених даних. Патерни доступу до даних змінюються в міру еволюції системи. Кількість клієнтів зростає у міру зростання популярності системи. Кількість збережених даних зазвичай також зростає в міру розвитку системи. Все це призводить до того, що старі рекорди і успіхи стають неактуальними, і потрібно заново проводити процес тонкого налаштування системи, а іноді й думати про зміну архітектури доступу до даних.

Як вже зазначалося, кожна фізична машина в будь-який момент часу може зламатися. Крім того, у будь-якій фізичній машини є межа продуктивності, яку вона може забезпечити. Ці дві обставини змушують об'єднувати машини в мережу і розглядати їх як розподілену базу даних. Розподілені бази даних змушують заново задуматися про всіх питаннях, які ми обговорювали для випадку однією фізичною машини: про моделі даних, протоколі доступу до даних і про гарантії, які забезпечуються у разі відмови обладнання.

Відомо кілька механізмів доступу до даних у БД NoSQL.

1) Restful інтерфейси. Це інтерфейс, схожий на основний протокол Інтернету – HTTP. У межах цього підходу передбачається, що кожний об'єкт, яким ми можемо маніпулювати, має свою унікальну адресу. Звертаючись за цією адресою, ми можемо запитувати, створювати, редагувати або знищувати зазначений об'єкт. При цьому на сервері не зберігається стан запиту, тобто кожний запит опрацьовується незалежно від інших запитів.

2) Мови запитів, відмінні від SQL.

• GQL – Sql-подібна мова для Google Bigtable;

• SPARQL – мова запитів Семантичного Веба;

• Gremlin – мова обходу графів;

• Sones Graph Query Language – мова запитів до Sones Graph.

3) API запити:

• Google Bigtable Datastore API;

• Neo4j Traversal API.

Отже, існування різних категорій баз даних NoSQL вимагає формального опису моделей даних, що ними опрацьовуються. Аналіз згаданих вище джерел, науково-популярних журналів і блогів дають змогу виділити такі вектори роботи з Великими даними:

• джерела появи Великих даних;

• апаратне забезпечення та інфраструктура;

• програмне забезпечення й способи зберігання;

• інформаційні технології (методи і засоби обробки даних);

• використання Великих даних, бізнес-аналіз.

2.2 Формальний опис структури Великих даних

Прикладом Великих даних є масив даних, що описує процеси функціонування регіону. Отже, у підсумку існує:

• набір сутностей: особи, місця, організації (фізичні, юридичні), дати, природні ресурси (річки, ліси, озера), рекреаційний фонд (історичні пам’ятки, санаторії), законодавчі акти та звіти;

• база даних описів сутностей: документи для інтелектуального аналізу даних, аналогічні терміни, словники даних, які дозволяють зв'язати деякі об'єкти.

Ґрунтуючись на цій інформації, ми повинні визначити, які сутності і в який спосіб пов’язані між собою. Формально поділимо усі об’єкти на такі категорії:

• сутності e ,

• характеристики f ,

• асоціації між сутностями e та характеристиками f . Наприклад:

• ім’я e згадується у документі f ,

• термін f з’явився у документі e . Нехай також визначено:

• множину сутностей E ;

• множину характеристик F ;

• для кожних e і f зазначено номер асоціацій між e і f як fe n , .

Загальна кількість сутностей визначається як E , загальна кількість характеристик є потужністю множини F : F . Також опишемо:

• для кожної характеристики f множину { nEefe , fe >∈= 0:)( } усіх асоційованих з f сутностей;

• для кожної сутності e множину f ef )( {f EFn , fe >∈= 0} усіх асоційованих з e характеристик.

2.3 Моделі асоціацій між сутностями та характеристиками для різних категорій NoSQL БД

Носій даних у моделі «ключ-значення» (інша назва – колонкова БД) описується формулою 2.1.

|  |  |
| --- | --- |
| KV = { < f, e>}, | (2.1) |

де f – ключ, який приймає унікальні значення у кожній парі, e – значення, що відповідає цьому ключеві.

Сигнатура моделі така наведена у формулі 2.2.

|  |  |
| --- | --- |
| O = < σπ >, | (2.2) |

де π – операція проекції за атрибутами (ключ або значення), σ – селекції атрибутів (вибір значення за ключем, ключів за значенням, ключів за значенням предків).

Прикладом СУБД колонкового типу є Cassandra. Для версійного розподіленого зберігання великих обсягів даних була спроектована модель, що використовується у системі BigTable компанії Google з такими характеристиками:

* не повна реляційна модель даних,
* підтримка динамічного контролю над розміщенням даних.

Основа моделі даних Bigtable проста: рядки, стовпці й тимчасові мітки. Описується формулою 2.3.

|  |  |
| --- | --- |
| BigTable = { <r,c,t> } | (2.3) |

Наприклад, у базі даних пошукової машини іменами рядків можуть слугувати адреси документів з Інтернету, а іменами стовпців – особливості цих документів (наприклад, зміст документа може зберігатися в стовпці «content:», а посилання на дочірні сторінки – в шпальтах «anchor:»). Інший приклад – карти Google, що складаються з мільярдів зображень, кожне з яких деталізує ту чи іншу географічну ділянку планети. У Bigtable карти Google структуруються так: кожному рядку відповідає один географічний сегмент, а стовпцями є зображення, з яких цей сегмент складається, у різних стовпцях зберігаються зображення з різною деталізацією.

Рядки Bigtable (їх максимальна довжина може досягати 64 кілобайти) теж важливі. Операція звернення до рядка є атомарною (це означає, що поки одна програма звертається до рядка, жодна інша не має права змінювати дані в сімействах стовпців цього рядка). А ще рядки зручно сортувати. У прикладі з URL документа, зробивши його запис реверсивним, легко впорядкувати всі рядки за іменем домена третього рівня.

Вміст сторінок в Інтернеті постійно змінюється. Щоб врахувати ці зміни, кожній копії даних, що зберігаються в стовпці, присвоюється тимчасова мітка (timestamp). У Bigtable тимчасовою міткою слугує 64-розрядне число, яким можна кодувати час і дату так, як це потрібно клієнтським програмам. Наприклад, timestamp для копій веб-сторінки в стовпці «contents:» є датою і часом створення цих копій. Використовуючи тимчасові мітки, додатки можуть задати в Bigtable пошук, наприклад, тільки найостанніших копій даних.

Отже, для будь-якої предметної області в сервісі Google можна створити власну карту даних Bigtable, що містить задану кількість рядків і унікальний для цієї предметної області набір сімейств стовпців. Повтори даних у стовпцях упорядковуються за значеннями тимчасових міток. Усе це вказує на повну відсутність підтримки властивостей ACID. Головною перевагою цього підходу є те, що таку базу неважко порізати на незалежні шматочки і розподілити по множині серверів. Відсортовані за алфавітом рядки діляться на діапазони, які називають «таблет» (tablet) – несамостійними таблицями. Оскільки рядки в кожному таблеті відсортовані за ключовим іменем, то клієнтським додаткам просто знайти потрібний таблет, а в ньому – потрібний рядок [4]. Для синхронізації таблетів використовують сервіс Chubby. Його роль в Bigtable можна порівняти з роллю транзакцій в звичайних СУБД. Для кожного таблет-сервера Chubby створює спеціальний chubby-файл. Завдяки цьому файлу Bigtable може визначити, які з серверів працездатні. Ще один chubby-файл містить посилання на розташування кореневого таблета (Roottablet) з даними про розташування усіх інших. Цей файл повідомляє майстру, який з серверів якими таблетами керує. Безумовно, використання сервісу Chubby в Bigtable якоюсь мірою виконує завдання підтримки несуперечності даних у розподіленому середовищі з безліччю реплік. Але несуперечливість буває різною. Bigtable стала першою спробою досягти балансу між продуктивністю системи, її масштабованістю і непротиріччям даних, що тут зберігаються [9]. Результатом стала підтримка так званої слабкої несуперечності, яка, в принципі, задовольняла вимоги більшості працюючих з Bigtable сервісів.

Носій моделі «об’єкт-документ» описується кортежами вигляду, який представлено у формулі 2.4.

|  |  |
| --- | --- |
|  | (2.4) |

де f0 – ідентифікатор документа, f1…fn – характеристики (атрибути) документа, e1…en – атомарні значення характеристик f1…fn, d1…dn – посилання на інші документи, di = e(fi).

Операції цієї моделі є об’єктні.

Операція визначення вузлів елемента:

|  |  |
| --- | --- |
|  | (2.5) |

де C – колекція документів odi

Операція визначення значень вузлів:

|  |  |
| --- | --- |
|  | (2.6) |

де еij – значення атрибутів fi.

Також визначено відношення над елементами носія. Відношення «елемент-елемент» визначаються між документами та колекцією:

|  |  |
| --- | --- |
|  | (2.7) |

Відношення «елемент-атрибут»:

|  |  |
| --- | --- |
|  | (2.8) |

Відношення «елемент-посилання»:

|  |  |
| --- | --- |
|  | (2.9) |

Відношення «елемент-дані» визначаються так:

|  |  |
| --- | --- |
|  | (2.10) |

Прикладами СУБД цього типу є MongoDB та CouchDB.

Графова модель даних подана як:

|  |  |
| --- | --- |
|  | (2.11) |

де ID – множина ідентифікаторів, вузлів графа; А — множина позначених спрямованих дуг (p,l,с), ,p,c ∈ ID , l — « рядок-мітка»; запис (р,l,с) означає, що між вузлами р та с є зв’язок l; z — функція, що відображає кожний вузол n ∈ ID в конкретне значення складеного або атомарного типу, z : n → v ; V — особливий кореневий вузол графа.

Структура XML-документа, що складається з вкладених елементів-тегів добре відома, її відмінність від розглянутої вище графової моделі полягає, в основному, у трактуванні тегів і міток: в графах мітки використовуються як позначення зв'язків між елементами схем даних, і мітки не потрібні для позначення елемента, а в XML документно-орієнтованій моделі потрібно, щоб кожний (нетекстовий) елемент даних мав ідентифікуючу ознаку. Також XML транслюється в структуру даних «дерево», що є частковим випадком графової моделі.

У графовій моделі XML для слабко структурованих даних необхідно використовувати спеціалізовані типи атрибутів, такі як ID, IDREF, IDREFS [5]. Зазначені типи дають змогу організувати зберігання перехресних посилань в XML -елементах виду (eid,vahie) (<ідентифікатор eлемента, значення>) і атрибутах виду < label, eid> (<мітка, значення>).

Існує кілька видів RDF-даних як графової моделі: RDF / XML, N3, Turtle, RDF / JSON. Опис ресурсів у вигляді RDF -набору даних – це трійка «суб'єкт»- «предикат»-«об'єкт», тобто для трійки:

• множини U (Universal Resource Identifier, URI, уніфікований ідентифікатор ресурсів) – елементи f,

• множини В {Black nodes, порожніх вузлів),

• множини L {Literal, RDF-літералів), , ∈∈ eLeB,

визначено набір (f, e(f), e), де f – «суб'єкт»; e(f) – «предикат»; e – «об'єкт».

Отже, великі дані поєднують дані, представлені у різних моделях даних. Для цього повинні існувати методи їх перетворення з мінімальною втратою даних.

2.4 Особливості NoSQL моделей збереження даних

* 1. Не використовується SQL

Нереляційні бази даних не використовують загальноприйнятий ANSI SQL DML, але вони намагаються використовувати query languages, які схожі на загальновідомий улюблений синтаксис.

* 1. Неструктуровані

Структура не реляційних NoSQL баз даних не регламентована (або слабо типизована) – у кожному рядку або документі можна добавити довільне поле без попереднього декларативної зміни структури усій таблиці. Таким чином, якщо з’являється необхідність змінити модель даних, то єдина достатня дія - відобразити зміну в коді програми.

Якщо ми змінюємо логіку додатку, значить ми очікуємо нове поле також і при читанні. Але, через відсутність схеми даних поле totalSum відсутня у інших вже існуючих об'єктів Order. У цій ситуації є два варіанти подальших дій. Перший - обійти всі документи і оновити це поле у всіх існуючих документах. В силу обсягів даних цей процес відбувається без будь-яких блокувань (порівняймо з командою alter table rename column), тому під час оновлення вже існуючі дані можуть зчитуватися іншими процесами. Тому другий варіант - перевірка в коді програми – неминучий.

А вже при повторній записи ми запишемо це поле в базу в новому форматі.

Приємний наслідок відсутності схеми - ефективність роботи з розрідженими (sparse) даними. Якщо в одному документі є поле date\_published, а в другому - ні, значить ніякого порожнього поля date\_published для другого створено не буде. Це, в принципі, логічно, але менш очевидний приклад - column-family NoSQL бази даних, в яких використовуються знайомі поняття таблиць / колонок. Однак у силу відсутності схеми, колонки не оголошуються декларативно і можуть мінятися / додаватися під час користувача сесії роботи з базою. Це дозволяє зокрема використовувати динамічні колонки для реалізації списків.

У неструктурованої схеми є свої недоліки - крім згаданих вище накладних витрат у коді програми при зміні моделі даних - відсутність всіляких обмежень з боку бази (not null, unique, check constraint і т.д.), плюс виникають додаткові труднощі в розумінні і контролі структури даних при паралельній роботі з базою різних проектів (відсутні будь-які словники на стороні бази). Втім, в умовах швидко мінливого сучасного світу така гнучкість є все-таки перевагою. Як приклад можна привести Твіттер, який років п'ять тому разом з твіттами зберігав лише трохи додаткової інформації (час, Twitter handle і ще кілька байтів метаінформації), проте зараз в додаток до самого повідомленням в базі зберігається ще кілька кілобайт метаданих.

* 1. Представлення даних у вигляді агрегату

На відміну від реляційної моделі, яка зберігає логічну бізнес-сутність додатки в різні фізичні таблиці в цілях нормалізації, NoSQL сховища оперують з цими сутностями як з цілісними об'єктами:

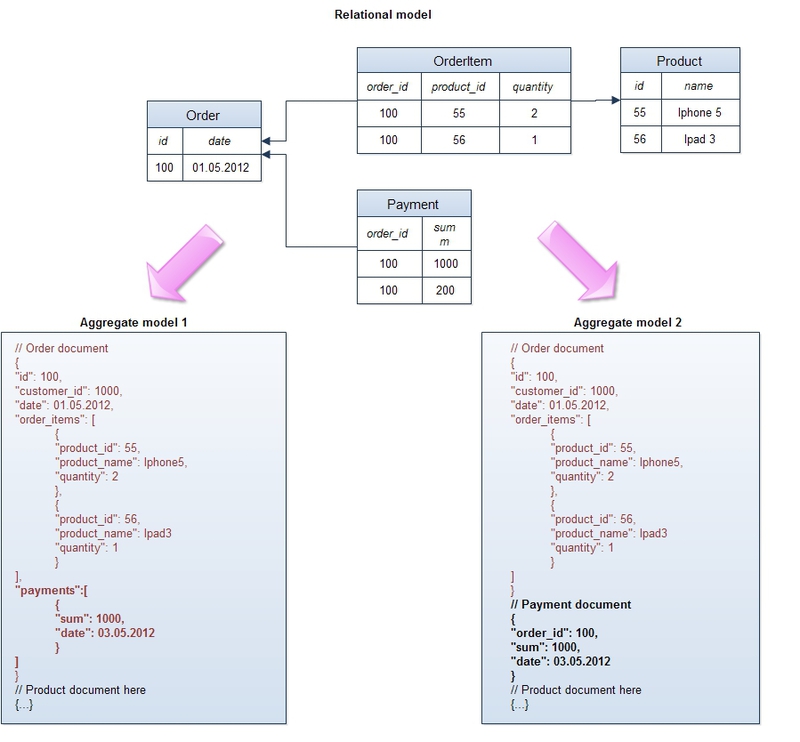


Рисунок 2.2 – Агрегатна модель представлення даних

У цьому прикладі продемонстровані агрегати для стандартної концептуальної реляційної моделі e-commerce "замовлення - позиції замовлення - платежі - продукт". В обох випадках замовлення об'єднується з позиціями в один логічний об'єкт, при цьому кожна позиція зберігає в собі посилання на продукт і деякі його атрибути, наприклад, назва (така денормалізація необхідна, щоб не запитувати об'єкт продукту при витяганні замовлення - головне правило розподілених систем - мінімум "Джоін" між об'єктами). В одному агрегаті платежі об'єднані із замовленням і є складовою частиною об'єкта, в іншому - винесені в окремий об'єкт. Цим демонструється головне правило проектування структури даних в NoSQL базах - вона повинна підкорятися вимогам програми та бути максимально оптимізованої під найбільш часті запити. Якщо платежі регулярно витягуються разом із замовленням - має сенс їх включати в загальний об'єкт, якщо ж багато запитів працюють тільки з платежами - значить, краще їх винести в окрему сутність.

Багато хто заперечить, зауваживши, що робота з великими, часто денормалізованими, об'єктами чревата численними проблемами при спробах довільних запитів до даних, коли запити не вкладаються в структуру агрегатів. Що, якщо ми використовуємо замовлення разом з позиціями і платежами за замовленням (так працює додаток), але бізнес просить нас порахувати, скільки одиниць певного продукту було продано минулого місяця? У цьому випадку замість сканування таблиці OrderItem (у разі реляційної моделі) нам доведеться витягувати замовлення цілком в NoSQL сховище, хоча більша частина цієї інформації нам буде не потрібна. На жаль, це компроміс, на який доводиться йти в розподіленої системі: ми не можемо проводити нормалізацію даних як у звичайній одно серверній системі, оскільки це створить необхідність об'єднання даних з різних вузлів і може призвести до значного уповільнення роботи бази.

Плюси та мінуси можна наведені у одній згрупованій таблиці 2.5.

Таблиця 2.5 – порівняння методів збереження даних у вигляді нормалізованих даних та у вигляді агрегатів.

|  |  |
| --- | --- |
| Нормалізація даних | Дані у вигляді агрегатів |
| * Цілісність інформації при обновленні (змінюємо запис у одній таблиці, а не деяких); * Орієнтованість на широкий спектр запитів до даних; * Неефективна у розподіленому середовищі; | * Оптимізація тільки під певний тип запитів; * Складність при обновленні ненормалізованих даних; * Кращий спосіб досягти більшої швидкості при читанні у розподіленому середовищі; |

Продовження таблиці 2.5

|  |  |
| --- | --- |
| * Повільна швидкість читання при використанні об’єднань (joins) * Невідповідність об’єктної моделі додатку фізичній структурі даних (вирішується за допомогою технічних рішень, які звуться ORM). | * Можливість зберігати фізичні об’єкти у такому виді, в якому з ними буде легше працювати та будувати логіку програмного додатку; * Рідна (native) підтримка атомарності на рівні записів. |

4) Слабкі ACID-властивості

Довгий час консистентность (consistency) даних була "священною коровою" для архітекторів і розробників. Всі реляційні бази забезпечували той чи інший рівень ізоляції - або за рахунок блокувань при зміні і блокуючого читання, або за рахунок undo-логів. З приходом величезних масивів інформації і розподілених систем стало ясно, що забезпечити для них транзакційність набору операцій з одного боку і отримати високу доступність і швидкий час відгуку з іншого - неможливо. Більше того, навіть оновлення одного запису не гарантує, що будь-який інший користувач моментально побачить зміни в системі, адже зміна може відбутися, наприклад, у Master Node, а репліка асинхронно скопіюється на слейв-ноду, з якою і працює інший користувач. У такому випадку він побачить результат через якийсь проміжок часу. Це називається eventual consistency і це те, на що йдуть зараз всі найбільші інтернет-компанії світу, включаючи Facebook і Amazon. Останні з гордістю заявляють, що максимальний інтервал, протягом якого користувач може бачити неконсистентне дані становлять не більше секунди. Приклад такої ситуації показаний на малюнку:

Насправді слабкі ACID властивості не означають, що їх немає взагалі. У більшості випадків додаток, що працює з реляційною базою даних, використовує транзакцію для зміни логічно пов'язаних об'єктів (замовлення - позиції замовлення), що необхідно, так як це різні таблиці. При правильному проектуванні моделі даних в NoSQL базі (агрегат представляє з себе замовлення разом з переліком пунктів замовлення) можна домогтися такого ж самого рівня ізоляції при зміні одного запису, що і в реляційній базі даних.

5) Розподілені системи, без спільно використовуваних ресурсів

Знову ж таки, це не стосується графових баз даних, структури яких за визначенням погано розноситься по віддаленим вузлам.

Це, можливо, головний лейтмотив розвитку NoSQL баз даних. З обвальним зростанням інформації у світі і необхідності її обробляти за розумний час постала проблема вертикальної масштабованості - зростання швидкості процесора зупинився на 3.5 Ггц, швидкість читання з диска також зростає тихими темпами, плюс ціна потужного сервера завжди більше сумарної ціни декількох простих серверів. У цій ситуації звичайні реляційні бази, навіть кластеризовані на масиві дисків, не здатні вирішити проблему швидкості, масштабованості і пропускної здатності. Єдиний вихід із ситуації - горизонтальне масштабування, коли декілька незалежних серверів з'єднуються швидкої мережею і кожен володіє / обробляє лише частину даних та / або тільки частина запитів на читання-оновлення. У такій архітектурі для підвищення потужності сховища (ємності, часу відгуку, пропускної спроможності) необхідно лише додати новий сервер в кластер - і все. Процедурами шардінга, реплікації, забезпеченням відмовостійкості (результат буде отриманий навіть якщо одна або кілька серверів перестали відповідати), перерозподілу даних у разі додавання вузла займається сама NoSQL база. Коротенько представлю основні властивості розподілених NoSQL баз:

а) Реплікація – копіювання даних на інші вузли при обновленні. Дозволяє як добитися великої масштабованості, так і підвищити доступність та збереженість даних. Прийнято розділяти на два виду:

- master-slave:

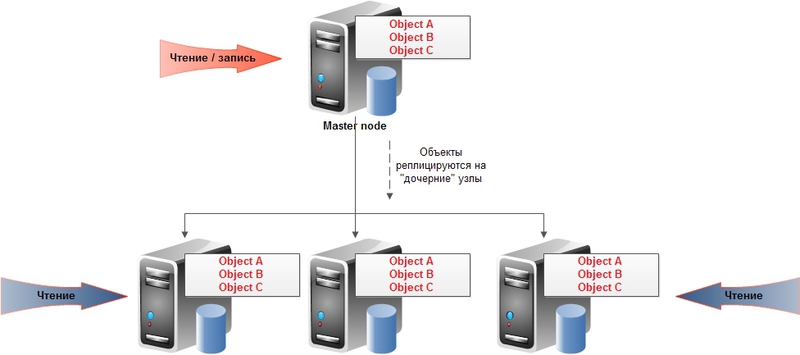


Рисунок 2.3 – Master-slave реплікація

- peer-to-peer:

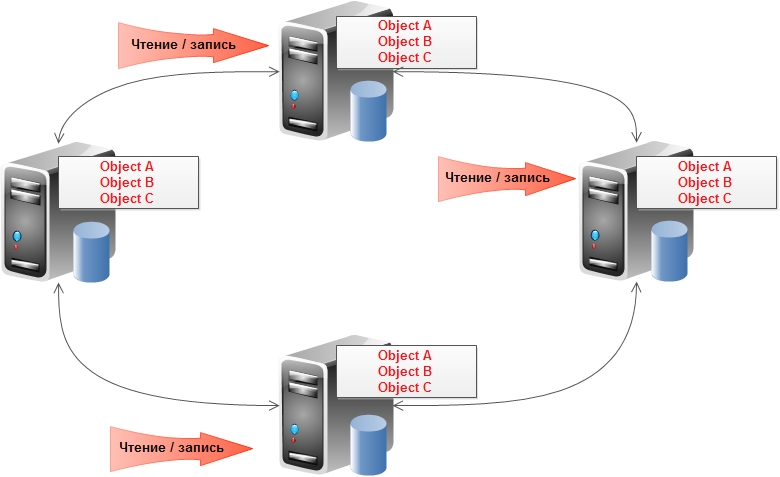


Рисунок 2.4 – Peer-to-peer реплікація

Перший тип припускає гарну масштабованість на читання (може відбуватися з будь-якого вузла), але не масштабований запис (тільки в майстер вузол). Також є тонкощі із забезпеченням постійної доступності (у разі падіння майстра або вручну, або автоматично на його місце призначається один з решти вузлів). Для другого типу реплікації передбачається, що всі вузли рівні і можуть обслуговувати як запити на читання, так і на запис.

Шардінг часто використовувався як "милиця" до реляційних баз даних з метою збільшення швидкості та пропускної здатності: користувальницький додаток партіціровало дані по декількох незалежних баз даних і при запиті відповідних даних користувачем зверталося до конкретної бази. У NoSQL базах даних шардінг, як і реплікація, проводяться автоматично самою базою і користувальницький додаток відокремлено від цих складних механізмів.

Наступні характеристики часто пов’язані з NoSQL базами даних:

* NoSQL бази даних розроблялися для управління великими обсягами даних, яким не обов'язково дотримуватися фіксованою схемою;
* Вони не можуть гарантувати повну ACID (атомарность, узгодженість, ізольованість, довговічність) гарантію;
* Вони мають розподілену відмовостійку архітектуру, можуть легко масштабуватися за рахунок додавання нових серверів і відмова одного сервера не вплине на роботу. Цей тип баз даних звичайно масштабується по горизонталі та використовується для управління великими обсягами даних, коли продуктивність у режимі реального часу є більш важливою, ніж консистенція даних.

NoSQL бази даних часто оптимізовані для операцій пошуку та додавання нового елементу. Вони можуть бути корисними при роботі з великою кількістю даних, які не потребують опису своєї структури. Для NoSQL систем важлива здатність зберігати та витягувати великі обсяги даних, а не відношення між цими елементами. Це особливо важливо для аналізу статистичних або швидкозростаючих у режимі реального часу елементів (наприклад повідомлення у соціальній мережі Twitter)[9].

У даній роботі основна увага зосереджена на найбільш популярних документо-орієнтованих нереляційних базах даних, таким як CouchDB, MongoDB, CassandraDB, подається їх огляд та порівняння продуктивності для операцій вставки, вибору та видалення об’єктів. Також буде приведено порівняння даних нереляційної БД з реляційною БД MS SQL Server.

CouchDB – документо-орієнтована система керування базами даних, яка не потребує опису схеми даних. На відміну від реляційних баз даних, CouchDB не зберігає дані та відношення у таблиці. Замість цього, кожна база даних представляє собою набір незалежних документів. Кожний документ зберігає свої власні та автономні схеми. Програмний додаток може отримати доступ до декількох баз даних. Документ метаданих зберігає інформацію стосовно ревізій, що дозволяє визначити будь-які відмінності, які, можливо, відбулися у той час, коли деякі бази даних були відключені.

Особливості CouchDB:

* цілісність бази даних забезпечується виключно на рівні окремих записів (але не на рівні відношень між ними);
* зв’язки між таблицями або записами принципово не підтримуються, відповідно операція об’єднання (JOIN) між таблицями не визначена;
* одночасно може бути запущено декілька потоків для читання бази даних та тільки один – для запису;
* представлення зберігаються безпосередньо у тій самій БД, у якій самі записи та їх індекси обновлюються безперервно.

Переваги CouchDB:

- відсутність блокування та, як наслідок, висока доступність;

- інкрементальний MapReduce;

- зберігання усіх даних у Json документах;

- REST інтерфейс – усі операції, включаючи вставку даних, у CouchDB здійснюється через HTTP;

- множинна реплікація – можливість використовувати необмежену кількість приладів, створюючи топології реплікації.

До недоліків CouchDB можна віднести:

* складність написання запитів на MapReduce та, як наслідок, невелика швидкість розробки програмного додатку;
* довгий час вставки об’єктів.

MongoDB - документо-орієнтована система керування базами даних, яка не потребує опису схеми даних.

За рахунок мінімізації семантики для роботи з транзакціями у MongoDB з’являється можливість рішення цілого ряду проблем, пов’язаних з недоліками продуктивності, причому горизонтальне масштабування стає простіше. Використовувана модель документів збереження даних (JSON/BSON) простіше кодується, простіше управляється, а внутрішнє угрупування релевантних даних забезпечує додатковий виграш у швидкодії.

Особливості MongoDB:

* документо-орієнтоване сховище з JSON-подібною схемою даних;
* повна підтримка індексів;
* ефективне збереження бінарних даних великих обсягів;
* підтримка журналу роботи операцій, модифікацій даних у БД;
* підтримка масштабування: асинхронна реплікація, набір реплік та шардінг;
* може працювати у відповідності до парадигми MapReduce.

До недоліків MongoDB можна віднести:

* за замовчуванням максимальний розмір об’єкта – 4 мегабайту;
* на 32-бітних машинах, максимальний розмір однієї бази даних – 2.5 гігабайту.

Cassandra – розподілена система управління базами даних, яка відноситься до класу NoSQL-систем та розрахована на створення високо масштабованих та надійних сховищ великих масивів даних, які представляються у вигляді хешу.

Cassandra використовує модель збереження даних на основі групи стовбців (ColumnFamily), що відрізняється від систем, подібних до memcachedb, які зберігають дані тільки у зв’язці ключ/значення. Cassandra відноситься до категорії сховищ, підвищено стійким до збоїв. При збої вузла, його функції підхоплюються іншими вузлами. Додавання нових вузлів у кластер та обновлення версії Cassandra здійснюється на льоту, без додаткового ручного втручання та пере конфігурації інших вузлів.

Особливості Cassandra:

* децентралізувати – кожен вузол у кластері має однакове значення. Не існує єдиної точки відмови. Дані розподіляються по кластеру (так, що кожний вузол має різні дані), але не існує хазяїна, таким чином, кожен вузол може обслуговувати будь-які запити;
* підтримка реплікації та декількох центрів обробки даних реплікацій;
* мова запитів CQL (Cassandra Query Language) – альтернатива SQL.

2.5 Парадигма MapReduce, BSON, JSON

Виникла технологія MapReduce з необхідності обробляти дані, які в оперативну пам'ять ну ніяк не влізуть. Та й на жорсткий диск, взагалі-то, теж: технологія націлена на обробку петабайт (10 ^ 15) даних. Обробляються дані, відповідно, на кластері. Даними є таблиці, що містять список записів виду (KEY, VALUE). Користувачеві досить задати функції обробки Map і Reduce (докладніше про них - трохи пізніше), і платформа сама подбає про сортування даних, запуску функцій обробки, повторному виконанні упалих транзакцій і багато чим ще.

Дві функції, які пропонується реалізувати користувачеві, такі:

* 1. Map - функція, що приймає на вхід одну запис виду (KEY, VALUE), і повертає по ній будь-яку кількість нових записів (KEY1, VALUE1), (KEY2, VALUE2), ...;
  2. Reduce - функція, що приймає на вхід всі записи з даними ключем. Ітеруючись по ним, вона так само може повертати будь-яку кількість нових записів.

Більш формально, функція працює за принципом {(KEY, VALUE1), (KEY, VALUE2), ...} → (KEY1, VAL1), (KEY2, VAL2), ...

Сутність даних функцій полягає у тому, що обробляти петабайт даних можна, грубо кажучи, тільки лінійними алгоритмами. Якщо алгоритм працює O (n ^ 2) - можна вважати, що він просто не працює.

Для потокової лінійної обробки дані можна міняти і перетворювати (Map), і можна зливати, комбінуючи отримані результати (Reduce). Ці операції легко паралелі на кластерах з будь-якого числа машин, тому і допускають такі вражаючі обсяги масштабування.

Програмна модель map/reduce була запозичена з функціонального програмування, хоча в реалізації Hadoop і має деякі семантичні відмінності від прототипу в функціональних мовами.

Як і в функціональних мовах, при використанні програмної моделі map / reduce:

* вхідні дані не змінюються;
* розробник кодує, що потрібно зробити, а не як потрібно зробити.

На січень 2012 року широко відомі наступні програмні реалізації моделі map / reduce:

* Google MapReduce - закрита реалізація від Google на C ++;
* CouchDB і MongoDB - реалізації для NoSQL баз даних;
* Hadoop MapReduce - відкрита реалізація на Java для Apache Hadoop.

Повний список проектів, що використовують програмні реалізації програмної моделі map / reduce можна знайти в джерелі [20].

Hadoop MapReduce - програмна модель (framework) виконання розподілених обчислень для великих обсягів даних в рамках парадигми map / reduce, що представляє собою набір Java-класів і виконуваних утиліт для створення та обробки завдань на паралельну обробку.

Основні концепції Hadoop MapReduce можна сформулювати як:

* обробка / обчислення великих об’ємів даних;
* масштабованість;
* автоматичне розпаралелювання завдань;
* робота на ненадійному обладнанні;
* автоматична обробка відмов виконання завдань.

Роботу Hadoop MapReduce можна поділити на наступні етапи:

* 1. **Input read**

Вхідні дані діляться на блоки даних зумовленого розміру (від 16 Мб до 128 Мб) - спліти (від англ. Split). MapReduce Framework закріплює за кожною функцією Map певний сплати.

* 1. Map

Кожна функція Map отримує на вхід список пар «ключ / значення» <k, v>, обробляє їх і на виході отримує нуль або більше пар <k ', v'>, що є проміжним результатом.

map(k,v)->[(k', v')]

де k '- у загальному випадку, довільний ключ, який не збігається з k.

Всі операції map() виконуються паралельно і не залежать від результатів роботи один одного. Кожна функція map() отримує на вхід свій унікальний набір даних, що не повторюється ні для якої іншої функції map().

Підбиваючи підсумки, категоризувати NoSQL систему можна за допомогою наступної таблиці (табл. 2.7).

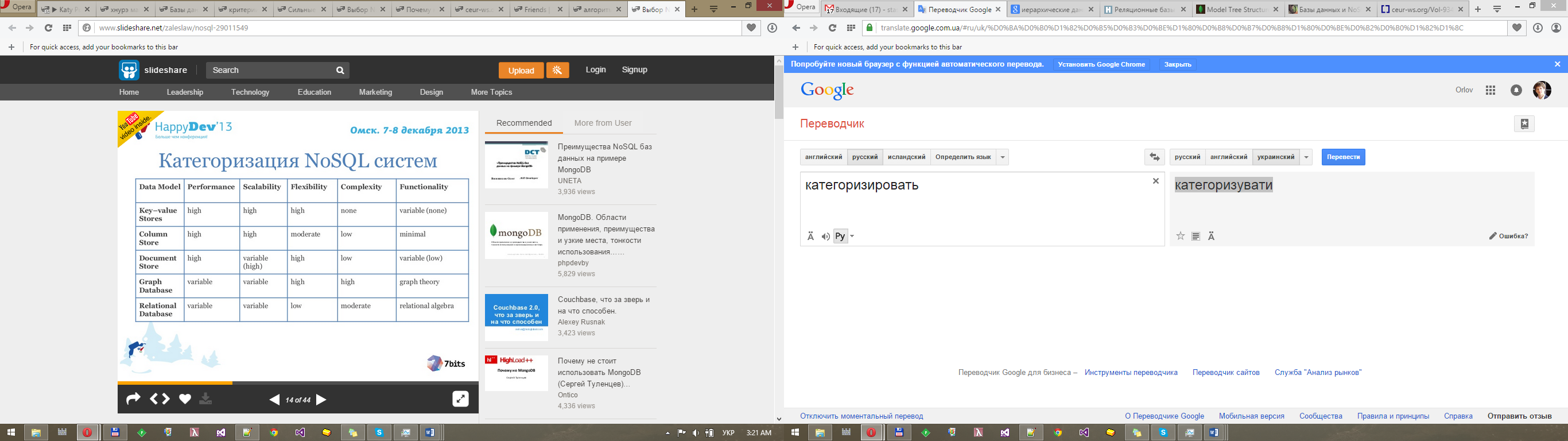


Рисунок 2.5 – Категоризування NoSQL систем

Основні переваги СУБД Raik: відмовостійкість, посилання на інші ключі та фільтри ключів, підсистема пошуку, можливість узгодженості та доступності на рівні окремого запиту.

Основні недоліки СУБД Raik: малі можливості запитів, відсутність ACID, неповна підтримка JavaScript, відсутність підтримки структур даних.

Основні переваги СУБД Mongo: повноцінна мова пошуку, зберігання складних денормалізованих документів, великий вибір індексів, реплікація даних.

Основні недоліки СУБД Mongo: ліміт на розмір результату (16 Мб), складні вибори, варто багато розмислювати над плануванням кластера, не гарантує правильність перевірки типів стовпців.

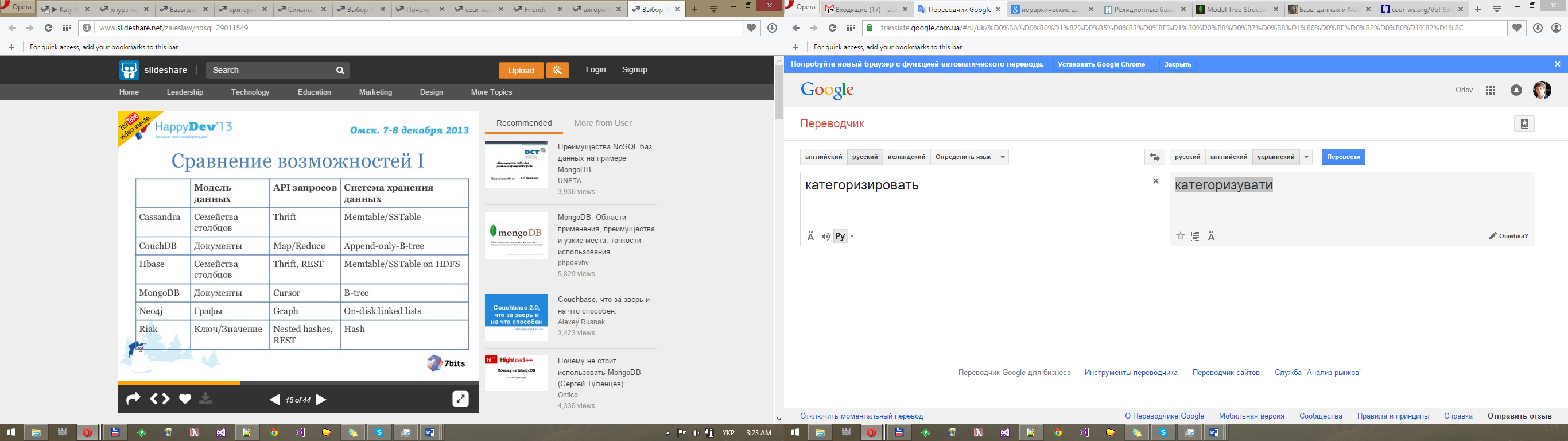


Рисунок 2.6 – Порівняння можливостей

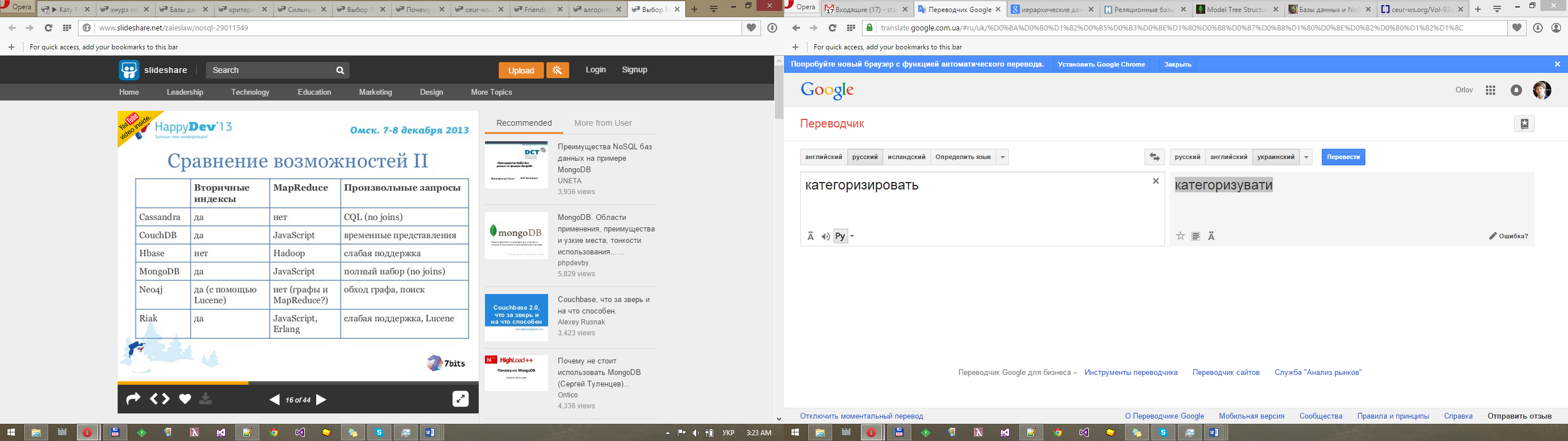


Рисунок 2.7 – Порівняння можливостей

У даному розділі розглянуто та описано реляційні та нереляційні моделі збереження даних, описано їхні математичні формальні моделі, описані їх основні особливості, переваги та недоліки. Також особливу увагу приділяється конкретним представникам кожного типу моделі збереження даних та їх особливостям.

3 ОПИС МЕТОДІВ ТА МОДЕЛЕЙ МАГІСТЕРСЬКОГО ДОСЛІДЖЕННЯ

3.1 Методи магістерського дослідження

Метод дослідження - це спосіб отримання збору, обробки або аналізу даних. Основним орієнтиром для вибору методу дослідження можуть служити його завдання. Саме завдання, поставлені перед магістерською роботою, визначають способи їх дозволу і вибирають відповідні методи дослідження.

У ході наукового дослідження планується провести і вирішити пізнавальні завдання, які будуть з'являтися і будуть поставлені в ході магістерського дослідження при порівнянні і проведенні аналізу реляційних і не реляційних СУБД. Дані завдання будуть спрямовані на виявлення, точний опис і детальне вивчення різних фактів, явищ і процесів. Тому для виконуваного дослідження в магістерській роботі буде використовуватися емпіричний метод дослідження.

Відповідно до цього методу планується отримувати нові знання на основі дослідження за допомогою опису, спостереження і експерименту.

Також для проведення дослідження емпіричним методом, необхідно буде попередньо провести глибоке теоретичне обгрунтування щодо умов проведення дослідження, збору емпіричних даних для експерименту, обробки отриманих результатів і формулювання висновків, виходячи з проведеного дослідження.

Так як для дослідження було обрано емпіричний метод, то планується провести дослідження, що складається з наступних етапів в наступній послідовності: збір інформації, спостереження явищ, аналіз, розробка гіпотези для пояснення явища, розробка теорії, що пояснюють феномен, заснований на припущеннях.

Необхідно провести аналіз існуючих досліджень у сфері порівняння СУБД і визначити основні положення, проблеми, які недостатньо розкриті в поточних дослідженнях. Також необхідно усвідомити потребу в усуненні дефіциту і встановити, для чого і як в подальшому можуть бути використані дані дослідження в області порівняння характеристик СУБД. Також тут необхідно визначити наявність випробовуваних точок, тобто визначити місця, фактори для проведення експерименту, які будуть розглянуті і перевірені в ході експерименту.

Планується провести збір інформації, визначити основні складові математичної моделі, грунтуючись на допустимих технічних характеристиках, обраних реляційних і не реляційних СУБД. Потім на основі зібраної інформації та отриманих факторах впливу математичної моделі, планується проведення експериментів і спостереження основних явища, виходячи з поставлених завдань магістерського дослідження. Тобто планується проведення експериментів на порівняння і спостереження основних характеристик, таких як швидкість виконання запитів на вибірку СУБД, витрати оперативної пам'яті СУБД і т.д.

В ході проведення збору інформації також необхідно буде визначити об'єкт і предмет дослідження. Забігаючи наперед, можна уточнити, що для даного дослідження об'єктом дослідження є задача визначення критеріїв вибору конкретного типу БД (Не реляційного або реляційного) залежно від типу розв'язуваної задачі, а предметом дослідження є оцінка та порівняльний аналіз швидкісних характеристик виконання запитів, характеристик витрачання пам'яті і структурних характеристик (зміни схеми бази даних) реляційних і не реляційних баз даних.

Мета повинна буде коротким і гранично точно, в смисловому плані виражати те основне, що має намір зробити в магістерській дослідницької роботу. Потім мета буде конкретизована в задачах наукового дослідження, тобто кожна задача повинна чітким формулюванням розкривати якісь сторони досліджуваної теми. У ході аналізу і роботи над цим блоком будуть сформульовані основні завдання дослідження магістерської роботи. Перше завдання буде пов'язана з виявленням, уточненням, поглибленням, металогіческіх обгрунтуванням сутності, структури досліджуваного об'єкта. Друга буде пов'язана з аналізом реального стану предмета дослідження. Третє завдання буде пов'язана з перетворенням предмета дослідження, тобто виявлення шляхів і засобів підвищення ефективності вдосконалення методів, способів здійснення запитів до баз даних з отриманням найкращої продуктивності. Четверте завдання буде пов'язана з дослідно-експериментальною перевіркою ефективності, тобто буде перевірена поточна продуктивність реляційних і не реляційних СУБД, оцінені основні слабкі місця, які знижують продуктивність і уповільнюють час виконання та запропоновано основні методи оптимізації про поліпшення знайдених проблем. Наприклад, у випадку великої кількості часу на вибірку даних можливе використання індексів в базі даних, для операцій вставки і оновлення даних, можливо заздалегідь видалення цих індексів і їх подальше відновлення після виконання операцій.

Для успішного проведення експерименту необхідно буде мати в наявності 2 бази даних, наповнені готовими даними достатнього обсягу, щоб мати можливість зробити основні часові виміри (щоб була достатня швидкість виконання). Також необхідно буде вирішити питання щодо платформи проведення експерименту (операційної системи). Для оцінки результатів проведення експериментів буде створено програмне забезпечення, що перевіряє і надає основні етапи, і результати дослідження.

У ході обробки буде використана математична модель обробки даних, тобто буде визначатися характер розподілу по всіх використовуваних параметрах дослідження.

Наприкінці будуть сформульовані висновки, що містять підсумки дослідження, тобто містять припущення про те, яка СУБД є більш придатною для вирішення поставленого завдання, які їхні сильні і слабкі сторони були виявлені в ході дослідження і які поради можна винести, виходячи з даного дослідження.

3.2 Модель магістерського дослідження

Допустимо ми маємо таблиці даних у декількох різних СУБД під керівництвом СУБД MS SQL та керівництвом Memcached, CouchDB, MongoDB, Cassandra, PostgreSQL. Для дослідження були вибрані по одному представнику кожного з типів баз даних: реляційні – MS SQL, СУБД «ключ-значення» - Memcached, СУБД колонкового типу – Cassandra, документо-орієнтована СУБД – MongoDB, CouchDB, СУБД на основі графів – GraphDB. Для дослідження будуть використані набір даних, розміром 100, 1000, 10000, 100000, 1000000 та випадковими значеннями даних. Програмні засоби, які реалізують алгоритми вимірювання головних показників, будуть запущені не менш ніж 100 разів кожен. Для дослідження використовувались C#, MS SQL 2014, Python, CouchDB, MongoDB, Cassandra, PostgreSQL. ПЗ, що виконує заміри для дослідження являє собою програмні додатки, написані за допомогою мов програмування C# та Python для операційної системи Windows. Існує можливість кешування даних засобами самого додатку. Необхідно оцінити наведені технології доступу до даних за характеристиками часу виконання вставки, вибірки, агрегації даних, знаходженню запису за ключовим полем (ID), видалення, пошуку за цілим значення більше заданого, пошуку за дробним значення меншим заданого, пошуком текстового значення (пошук підстроки), затраченої пам’яті, процесорному навантаженню.

В якості алгоритму для проведення досліджень необхідно вибрати простий алгоритм, складений з СRUD операцій та випадково створених даних. Апаратне забезпечення у даному випадку не має великого значення, оскільки основний інтерес являє собою відносна продуктивність та часові показники зазначених раніше досліджень.

Альтернативне дослідження засвідчило, проведене шляхом виконання програмних засобів засвідчило, що для даної задачі наявність або відсутність додаткового рівня програмного забезпечення (веб-серверу) не є принциповим, оскільки веб-сервер у даному випадку забезпечує тільки передачу даних між рівнем відображення та рівнем бізнес-логіки, що не є відносно обчислювальною складною задачею та займає відносно мале значення часу виконання по відношенню до виконання алгоритмів, реалізуючих основну логіку вирішення поставленої задачі.

Часові заміри та заміри стосовно використаної пам’яті для зберігання та обробки даних планується робити для декількох рівнів обсягів даних: для маленьких, для середніх та великих та порівняти результати на кожному з характеристик.

Як зазначалося раніше планується використовувати математичний метод для аналізу та порівняння отриманих результатів та визначення найкращого підходу використання СУБД у поєднанні з відповідним прикладним додатком, написаним на певній мові програмування, яка підібрана спеціально для найкращого поєднання під конкретну СУБД.

Для наукового дослідження планується використати функціональну математичну модель типу чорного ящика. Дана модель не використовує представлення внутрішню побудову та відображає лише поведінку (функціонування) об'єкта, що сприймається зовні. У їх граничному вираженні вони називаються також моделями «чорного ящика». Можливі також комбіновані типи моделей, які іноді називають моделями «сірого ящика».

Модель чорного ящику містить механізм для опису явища, хоча цей механізм недостатньо переконливий, не може бути достатньо підтверджений наявними даними або погано узгоджується з наявними теоріями і накопиченим знанням про об'єкт. Тому моделі чорного ящику мають статус тимчасових рішень. Вважається, що відповідь все ще невідома, і необхідно продовжити пошук «істинних механізмів».

Роль моделі в дослідженні може змінюватися з часом, може статися так, що нові дані і теорії підтвердять феноменологічні моделі і ті будуть підвищені до статусу гіпотези. Аналогічно нове знання може поступово прийти в протиріччя з моделями-гіпотезами першого типу, і ті можуть бути переведені до другого. Так, кваркова модель поступово переходить в розряд гіпотез; атомізм у фізиці виник як тимчасове рішення, але з ходом історії перейшов в перший тип. А ось моделі ефіру пройшли шлях від типу 1 до типу 2, а зараз знаходяться поза науки.

Ідея спрощення дуже популярна при побудові моделей. Але спрощення буває різним. Пайерлс виділяє три типи спрощень в моделюванні.

Для вирішення питань або проблем о необхідності вибору відповідного типу СУБД, необхідно проводити регулярний збір інформації про поточний стан змін та останніх модифікацій до відповідного типу СУБД, тобто виправлення якихось помилок або удосконалення якихось алгоритмів, наприклад, удосконалення швидкості вибору даних з таблиць або взагалі проведення операцій роботи з цими таблицями.

Другий критерій, якій необхідно брати до уваги це цільове направлення конкретного типу СУБД для вирішення тих чи інших задач. Необхідно приймати до уваги и звертати увагу на те, що взагалі необхідно керуватися поставленою для вирішення задачею. А потім вже відштовхуватися у тому чи іншому напрямку для вирішення питання порівняння та вибору СУБД.

Вибір відповідного типу СУБД - це початкова архітектура Вашої майбутньої системи та має дуже великий внесок у майбутній розвиток та масштабування, тому необхідно дуже відповідально відноситися до цього етапу.

Результуючими значеннями функції дослідження будуть час отримання даних при виконанні запита до бази даних та кількість необхідної оперативної пам’яті для виконання запитів та отримання даних, а також процесорне навантаження при виконанні запитів. Дані параметри будуть залежати від наступних складових: кількість даних (N), що оброблюються запитом, кількість наповнених даних у СУБД (F), тип СУБД (реляційна або нереляційна) (T), використання оптимізуючих елементів (наприклад, індексів бази даних) (O).

Математична модель являє собою сукупність трьох рівнянь, вихідними значеннями яких будуть швидкість виконання запиту Yt, завантаження оперативною пам’яті Ym та процесорне навантаження Yl.

Тоді математичну модель можливо представити у наступному вигляді сукупності рівнянь:

Yt = ft (N, F, T, O)

Ym = fm (N, F, T, O)

Yl = fl (N, F, T, O), де

N – фактор впливу, що визначає кількість даних, які оброблюються запитом,

F – фактор впливу, що визначає наповненість даних у СУБД,

T – фактор впливу, що визначає тип СУБД,

O - фактор впливу, що визначає використання оптимізуючих елементів.

Можливість залежностей між цими факторами виключена. Ці фактори є незалежними та впливають лише ззовні моделі на її обробку та кінцевий результат.

Необхідно буде провести експерименти за кожним з рівнянь до кожного типу СУБД та визначити фактор, що має найбільший вплив на результат.

Для пошуку оптимальних значень факторів впливу, для вирішення задачі оптимізації можна запропонувати наступні варіанти:

• повний перебір всіх можливих комбінацій значень факторів впливу;

• випадковий вибір деякого числа комбінацій і подальший вибір найкращого варіанта;

• аналітичне дослідження системи;

• застосування спеціалізованих програмних засобів;

• використання математичних моделей.

За формальною класифікацією ця модель лінійна, детермінична, динамічна, зосереджена, безперервна. У процесі її побудови ми зробили безліч припущень, які в реальності можуть не виконуватися.

По відношенню до реальності це, найчастіше, модель типу 4 спрощення («опустимо для ясності деякі деталі»), оскільки опущені деякі суттєві універсальні особливості. У деякому наближенні, така модель досить добре описує реальну систему, оскільки відкинуті фактори роблять пренебрежимо малий вплив на її поведінку. Однак модель можна уточнити, взявши до уваги якісь додаткові чинники. Це призведе до нової моделі, з більш широкою (хоча і знову обмеженою) областю застосовності.

Втім, при уточненні моделі складність її математичного дослідження може істотно зрости і зробити модель фактично марною. Найчастіше простіша модель дозволяє краще й глибше дослідити реальну систему, чим складніша (і, формально, «більш правильна»).

Втім, даний методи дослідження носять не зовсім точний характер, так як вони не враховують такі важливі параметри, як операційна система робочої станції (СУБД на Windows і Linux- подібних системах можуть продемонструвати абсолютно різні показники, якість заліза, тобто на скільки потужний сервер, на якому встановлена СУБД, також дане дослідження не враховує можливе поєднання обраного типу СУБД з конкретним типом фреймворками, в поєднанні з яким, використовуючи всі переваги фреймворками і технічні переваги СУБД можна домогтися поліпшення і значно поліпшити вплив факторів і власне скорегувати результат функції моделі).

Враховуючи отримані дані з результатів експерименту, необхідно буде визначити тип СУБД, що має найкращі показники серед запропонованих факторів. А також запропонувати метод покращення результатів для інших типів СУБД, що мали гірші результати за певним фактором впливу.

Структуру таблиці (об’єкта), над яким проводилися операції обробки даних та аналізу рівнянь моделі модна представити у наступному вигляді (Рисунок 3.1):

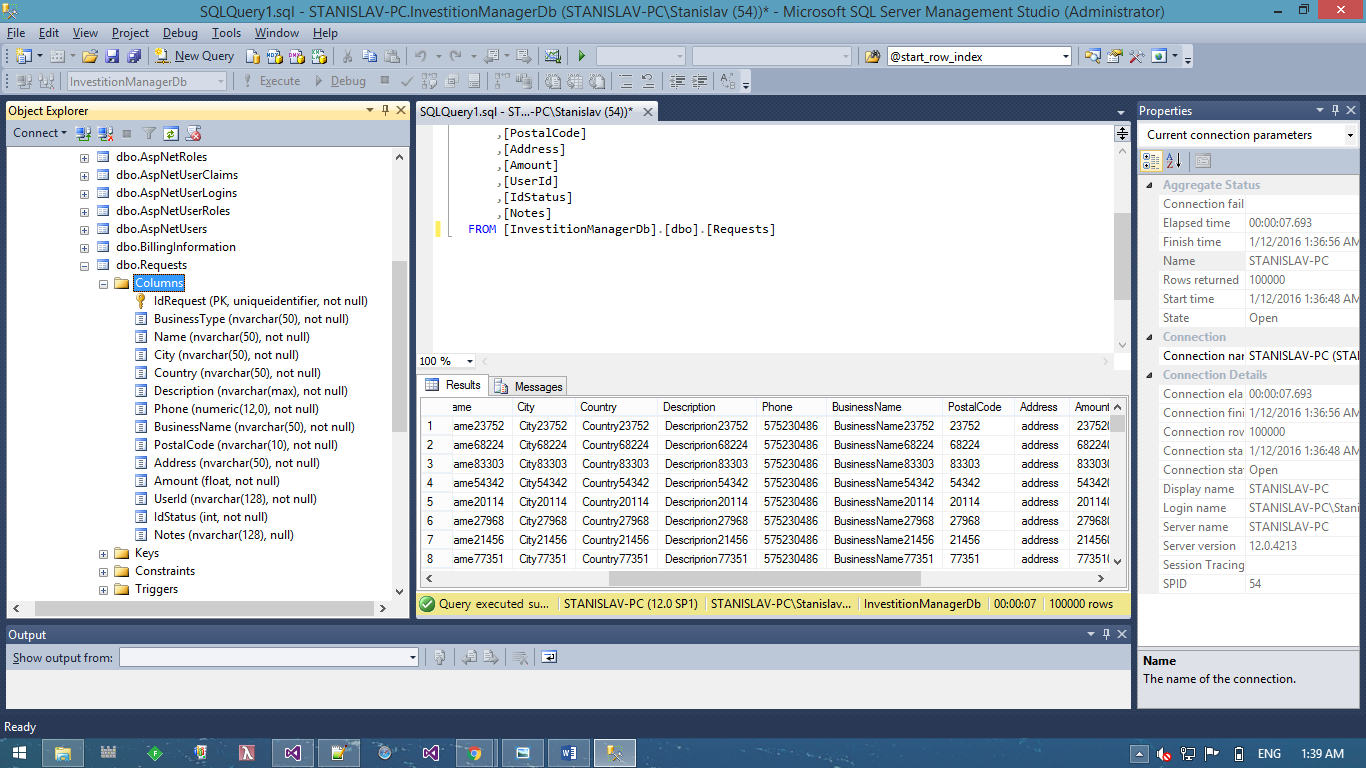


Рисунок 3.1 – Структура дослідницької таблиці

Таблиця поєднує у собі невелику кількість даних різного формату та типів, щоб зробити дослідження максимальне наближеним до реального проекту та реальних операцій бізнес логіки.

У даному розділі розглянуто основні математичні моделі та основні методи, які будуть застосовані у даній магістерській роботі відповідно до поставленої мети. Також наведено основну структуру таблиці, яка буде застосовуватися для проведення експериментів. Отриману математичну модель проаналізовано відповідно особливостей, а також розроблено методи удосконалення предмету дослідження на основі дослідження формальної математичної моделі.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ

Для написання цього продукту використовувалась мова програмування C# 4.5 та технологія .Net Framework 4.5, яка є дуже зручна для написання прикладних програм та веб-сайтів та має дуже простий та зручний синтаксис, що значно полегшує розробку та підтримку програмного продукту, а також мова програмування Python, що є дуже зручною для з нереляційними базами даних (які поширюються під ліцензією Open Source).

Даний проект являє собою додаток Windows без графічного інтерфейсу, щоб знизити додаткові витрати (на відкриття, закриття вікна та ін.) при виконанні замірів.

Даний проект складається з 4 частин, кожна з яких відповідає вибраній базі даних.

4.1 Реляційна MS SQL база даних

При виконанні вставки 1000 елементів у реляційну базу даних випадково генерувалися ці 1000 записів та у ході виконання циклу послідовно вставлялися у базу даних. При виконанні цієї функції замірювався час до початку та після виконання запитів, який використовувася для порівняння.

Також проводилися тести на вставку певної кількості даних у реляційну базу даних MS SQL Server. У даному тесті генерувалася вже необхідна кількість даних, яку потрібно було вставити до бази даних, виконувалася операція та замірювався час та інші показники.

Вибірка елементів (значення більше заданого параметру) у MS SQL Server наведена на рисунку 4.3. У даному тесті вимірювалися досліджуванні параметри при виконанні агрегатних функцій у базі даних. У якості агрегатних функцій використовувалися функції пошуку максимального та середнього значення.

Виконувана функція (максимум, середнє значення) визначалася вхідним параметром до процедури.

4.2 Сховище ключ-значення Redis

У якості бібліотеки доступу та взаємодії з Redis використовувалася бібліотека ServiceStack.Redis. Ця бібліотека є сумісною з .NetFramework та призначена для полегшення доступу до сховища Redis.

Сховище Redis підтримає два види видалення – видалення об’єкту за ідентифікатором та видалення самого об’єкту з колекції. У даному випадку розглядався другий варіант функції.

4.3 Розподілене сховище Cassandra

Для роботи з базою даних Cassandra у середовищі .Net використовувалася спеціальна бібліотека “CassandraCHarpDriver”. Ця бібліотека являє собою ORM для доступу до бази даних за допомогою технології .Net, який дозволяє використовувати SQL-мову запитів. Тобто цей драйвер дозволяє трансформувати мову запитів SQL у мову запитів CQL, яка підтримується внутрішнім середовищем СУБД Cassandra.

Програмний код пошуку об’єктів у середовищі Cassandra відповідно заданого ідентифікатору наведено на рисунку 4.7. Треба звернути увагу, що для використання наведеної ORM необхідно створити та звертатися до змінної session.

У цій функції виконується знаходження підстроки у великому (з точки зору розміру) полі таблиці.

СУБД Cassandra не має підтримки вхідних параметрів запиту, тому необхідно повністю формувати запит у програмному коді.

Функція агрегації не підтримується на рівні бази даних СУБД Cassandra, а підтримується тільки на рівні ORM, використаної для доступу до бази даних.

У даному розділі розглянуто програмну реалізацію основних методів, використаних для отримання кількісних характеристик згідно поставленої мети дослідження.

5 ПРОВЕДЕННЯ ЕКСПЕРИМЕНТУ З ПОРІВНЯННЯ БАЗ ДАНИХ

У даному розділі зазначені основні результати з проведення експерименті згідно поставленої задачі магістерської роботи. Узагальнені результати вставки об’єктів можна спостерігати на табл. 5.1.

Таблиця 5.1 - Вставка елементів (у секундах)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| БД | Кількість записів в БД | | | | | | |
| 0 | 1000 | 10000 | 100000 | 1000000 | 2000000 | 3000000 |
| NoSQL | 0,000115 | 0,000195 | 0,000099 | 0,000097 | 0,000102 | 0,000097 | 0,000094 |
| SQL | 0,000237 | 0,000225 | 0,000184 | 0,000162 | 0,000209 | 0,000163 | 0,000188 |

З отриманих результатів можна зробити висновок, що швидше виконується вставка об’єктів в нереляційну СУБД (NoSQL), що швидше за все обумовлено перевіркою, чи не порушує цілісність значення, що вставляється, а також додатковими затратами, пов’язаними з транзакціями при виконанні запита у реляційній СУБД.

Розглянемо також для порівняння швидкісні характеристики запитів на вибір об’єктів. У рамках даного тесту з бази даних будуть вибиратися усі записи, значення параметра яких більше за деяке значення. Результати можна побачити у таблиці 5.2.

Таблиця 5.2 – Вибір елементів з бази даних

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| БД |  | Кількість записів в БД | | | | | | |
|  | 0 | 1000 | 10000 | 100000 | 1000000 | 2000000 | 3000000 |
| NoSQL |  | 0,004375 | 0,00428 | 0,006966 | 0,050004 | 0,558619 | 1,064267 | 1,617517 |
| SQL |  | 0,011873 | 0,017049 | 0,066076 | 0,505179 | 5,321742 | 10,626587 | 15,98035 |

Як видно з результатів, при рості об’єму даних з операцією вибору елементів краще за всіх впорається нереляційна NoSQL СУБД, що пояснюється специфічним форматом збереження даних, який краще підходить для операцій вибору над великими об’ємами даних.

У даному експерименті порівнюється продуктивність основних операцій, таких як вставка, видалення та вибірка. У якості об’єкту для зберігання та обробки були взяті записи з фіксованим набором полів основних типів даних (цілочисельних, з плаваючою точкою, тимчасові, строкові). Для порівняння, експеримент проводився також на базі даних Redis, яка використовувалась як сховище ключ-значення з фіксованим набором полів. При проведені тестів використовувалася мова програмування Python та бібліотеки couchdb, pymogo, cql та psycopg2.

Операції вставки, видалення, вибірки проводилися при різній наповненості бази даних: 0, 1000, 10 000, 100 000, 1 000 000, 2 000 000 та 3 000 000 рядків даних. Враховуючи, що OrientDb показала низьку швидкість вставки елементів, то її тестування закінчилося на відмітці 10 000 елементів.

У ході проведення експерименту зі вставки елементів у базу даних вставлялося 1000 елементів та вимірювалася середня швидкість вставки елементів.

Результати вимірів наведені у таблиці 5.3.

Таблиця 5.3 - Час вставки 1000 елементів у базу даних (у секундах)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| БД | Кількість записів у Базі даних | | | | | | |
| 0 | 1000 | 10000 | 100000 | 1000000 | 2000000 | 3000000 |
| Cassandra | 0,000752 | 0,000315 | 0,00087 | 0,000864 | 0,000487 | 0,00093 | 0,000387 |
| Mongodb | 0,000115 | 0,000195 | 0,000099 | 0,000097 | 0,000102 | 0,000097 | 0,000094 |
| Redis | 0,000237 | 0,000225 | 0,000184 | 0,000162 | 0,000209 | 0,000163 | 0,000188 |
| OrientDb | 0,082746 | 0,082403 | 0,082367 |  |  |  |  |
| MS SQL | 1,1 | 40 | 0,0001 | 224 | 7900 |  |  |

З результатів можна побачити, що швидше за все вставка об’єктів виконується у базі даних Mongodb. СУБД MS SQL показала менший час, що, скоріше за все, обумовлено перевіркою, не порушує лі цілісність значення, що вставляється, а також накладними витратами, пов’язаними з транзакціями при виконанні запиту. У СУБД Cassandra стовпчиковий формат зберігання даних, тобто у пам’яті послідовно зберігаються не кортежі, а атрибути об’єктів, що у свою чергу приводить до додаткових накладних витрат при розміщенні об’єкта у пам’яті. СУБД OrientDb написана на мові програмування erlang, що не найкращим чином позначається на її продуктивності. Дана СУБД зберігає усі записи у вигляді строк у JSON форматі, що також уповільнює роботу з даними, тому що операції роботи зі строками вельми затратні. Хоча дані у СУБД Mongodb також представлені у JSON форматі, але для зберігання та обробки використовується формат BSON (Binary Json), який є бінарною формою представлення простих типів даних.

Результати експерименту вставки різної кількості даних до різних типів СУБД можна побачити у таблиці 5.4.

Таблиця 5.4 - Час вставки певної кількості даних до СУБД (у секундах)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| БД | Кількість записів, що вставлялися до бази даних | | | |
| 1000 | 10000 | 100000 | 1000000 |
| Cassandra | 0,000752 | 0,000819 | 0,000917 | 0,001082 |
| Mongodb | 0,000115 | 0,000421 | 0,000412 | 0,00521 |
| Redis | 0,000237 | 0,009123 | 0,012313 | 5,1212 |
| OrientDb | 0,082746 | 0,08342 |  |  |
| MS SQL | 1,1 | 13хв 47 сек | 137хв 43 сек | 233хв 24сек |

У рамках тесту вибірки елементів вибираються усі записи, значення вибраного параметру яких більше деякого значення. Даний запит реалізується у розглянутих СУБД по-різному. У OrientDb для цього використовується MapReduce функції. MongoDb володіє вбудованими операторами, які дозволяють накладати на збережені дані прості фільтри. У Cassandra та Redis даний запит виконується за допомогою мов запитів: CQL та SQL відповідно.

Результати вимірів наведені у таблиці 5.5.

Таблиця 5.5 – Час вибірки записів, більше вибраного параметру при різній наповненості бази даних.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| БД | Кількість записів у БД | | | | | | |
| 0 | 1000 | 10000 | 100000 | 1000000 | 2000000 | 3000000 |
| Cassandra | 0,047501 | 0,077999 | 0,387682 | 0,833015 | 0,920113 | 0,912034 | 0,917035 |
| Mongodb | 0,004375 | 0,00428 | 0,006966 | 0,050004 | 0,558619 | 1,064267 | 1,617517 |
| Redis | 0,011873 | 0,017049 | 0,066076 | 0,505179 | 5,321742 | 10,626587 | 15,98035 |
| OrientDb | 0,461459 | 0,439767 | 2,06743 |  |  |  |  |
| MS SQL | 0,54513 | 1,8342 | 10,34232 | 25,2121 | 65,04 | 1482 | 32573 |

З результатів можна побачити, що при рості об‘єму даних з операцією вибірки краще за всіх справляється СУБД Cassandra, що пояснюється поколоночним форматом зберігання даних, який, за словами дослідників краще за всіх підходить для операцій вибірки над великими об‘ємами даних.

У рамках тесту агрегації даних знаходиться середнє значення деякого поля, а також максимальне значення іншого поля. Даний запит у не реляційних СУБД виконується за допомогою MapReduce функції. У СУБД Cassandra існує підтримка MapReduce, але на рівні бібліотек мов програмування та технологій, які використовуються для доступу до даної СУБД. Тому у даному експерименті тести з Cassandra не проводилися. Результати наведені у таблиці 5.6.

Таблиця 5.6 – Час знаходження середнього значення

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| БД | Кількість даних | | | | | | |
| 0 | 1000 | 10000 | 100000 | 1000000 | 2000000 | 3000000 |
| Cassandra |  |  |  |  |  |  |  |
| Mongodb | 0,02547 | 0,044817 | 0,212003 | 1,905642 | 18,993331 | 37,344862 | 55,99169 |
| Redis | 0,002604 | 0,001927 | 0,007669 | 0,064599 | 0,672475 | 1,351388 | 2,031556 |
| OrientDb | 0,118381 | 0,156069 | 0,48995 |  |  |  |  |
| MS SQL | 0,46397 | 1,12321 | 10,45 | 143 | 368 | 682 | 782 |

З результатів можна побачити, що краще за усіх з задачею агрегації даних справляється СУБД Redis. Це пов‘язано з тим, що couchdb та mongodb використовуються парадигму MapReducе, яка полягає у тому, що над кожним об‘єктом у базі даних виконується деяка програма (Map-функція), результат якою поступає на вхід іншій програмі (Reduce-функції). У реляційній базі даних при агрегації виконуються деякі операції, на вхід яких подаються усі необхідні дані, що прискорює знаходження результату, але представляє менші можливості для розпаралелювання запиту, на відміну від парадигми MapReduce.

У наступному тесті знаходилося максимальне значення деякого поля. У нереляційних базах для цього використовувался функція MapReduce. Але вона не підтримується у базі даних Cassandra, тому у цьому тесті вимірювання для бази даних Cassandra не виконувалися. Результати вимірювання наведені у таблиці 5.7.

Таблиця 5.7 – Час знаходження максимального значення (у секундах)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| БД | Кількість записів у БД | | | | | | |
| 0 | 1000 | 10000 | 100000 | 1000000 | 2000000 | 3000000 |
| Cassandra |  |  |  |  |  |  |  |
| Mongodb | 0,025221 | 0,045554 | 0,219112 | 1,976588 | 20,486356 | 38,551766 | 58,22219 |
| Redis | 0,001017 | 0,000897 | 0,00369 | 0,028555 | 0,285282 | 0,576101 | 0,864792 |
| OrientDb | 0,126301 | 0,160708 | 0,52951 |  |  |  |  |
| MS SQL | 0,35893 | 1,02543 | 9,8734 | 125 | 255 | 579 | 801 |

У рамках даного тесту визначається середнє значення часу доступу до елементу бази даних за його ідентифікатором. Результати наведені у таблиці 5.8.

Таблиця 5.8 – Час доступу до елементу за ідентифікатором

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| БД | Кількість записів у БД | | | | | | |
| 0 | 1000 | 10000 | 100000 | 1000000 | 2000000 | 3000000 |
| Cassandra | 0,001277 | 0,00135 | 0,00101 | 0,002057 | 0,0064650 | 0,002325 | 0,002213 |
| Mongo | 0,000457 | 0,002228 | 0,006869 | 0,05232 | 0,571589 | 1,098512 | 1,699814 |
| Redis | 0,000437 | 0,000708 | 0,001662 | 0,009389 | 0,095481 | 0,19627 | 0,296314 |
| OrientDb | 0,040234 | 0,040194 | 0,040514 |  |  |  |  |
| MS SQL | 0,03212 | 0,0454 | 1,423 | 1,4546 | 1,5545 | 1,676 | 2,89435 |

З результатів видно, що найкраще з цим завданням справляється СУБД Cassandra, що досить дивно, т.к. стовпчиковий формат збереження даних добре підходить для вибору великих об’ємів даних, а не одиничного об’єкту. Швидше за все, результати обумовлені тим, що СУБД Cassandra, так само як і СУБД Redis, за замовчуванням використовує індекси на первинні ключі об’єктів.

У рамках тесту видалення елементів визначається середній час видалення одного запису з БД. Варто відзначити, що серед розглянутих СУБД видалити декілька записів (наприклад усі записи, значення параметру яких менше деякого значення) за один запит можуть лише СУБД MongoDb та Redis. Результати наведені у таблиці 5.9 та на рисунку 5.1.

Таблиця 5.9 – Час видалення елементів з БД (у секундах)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| БД | Кількість записів у БД | | | | | | |
| 0 | 1000 | 10000 | 100000 | 1000000 | 2000000 | 3000000 |
| Cassandra | 0,0005 | 0,000616 | 0,000638 | 0,000878 | 0,001048 | 0,000891 | 0,000848 |
| Mongo | 0,000056 | 0,000066 | 0,000056 | 0,000057 | 0,000077 | 0,000068 | 0,000054 |
| Redis | 0,000395 | 0,000706 | 0,00122 | 0,009146 | 0,095989 | 0,196652 | 0,296522 |
| OrientDb | 0,044951 | 0,045411 | 0,048183 |  |  |  |  |
| MS SQL | 0,000001 | 0,00005 |  |  |  |  |  |

Далі проводився експеримент, який полягав у пошуку по полям, більшого заданого параметру (число у діапазоні від 1000 до 10000000). Для проведення експерименту був введений додатковий атрибут – швидкість (кількість оброблених записів у секунду). Результати на ведені у таблиці 5.10.

Також проводилися експерименти стосовно процесорного навантаження та витраченої оперативної пам’яті. Результати експериментів наведені у таблицях 5.11 та 5.12 відповідно.

Таблиця 5.10 - Швидкість пошуку записів певного типу при наповненості бази даних у 1000000 записів, більше заданого значення або пошук входження рядка

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| БД | Тип поля, за яким проводиться пошук | | | |
| Id | Float | String | Text |
| Cassandra | 4892 | 264 | 479 | 25 |
| MongoDB | 47697 | 2249 | 5712 | 30 |
| Redis | - (виконання призводило до process terminated, налаштування не допомогли) | 1629 | 1254 | 20 |
| OrientDb | 517 | 112 | 57 | 24 |
| MS SQL | 2234 | 1567 | 2545 | 24 |

Таблиця 5.11 - Процесорне навантаження при виконанні пошуку записів певного типу при наповненості бази даних у 1000000 записів, більше заданого значення або пошук входження рядка

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| БД | Тип поля, за яким проводиться пошук | | | |
| Id | Float | String | Text |
| Cassandra | 24% | 12% | 24% | 35% |
| MongoDB | 31% | 34% | 35% | 40% |
| Redis | 5% | 12% | 19% | 25% |
| OrientDb | 35% | 23% | 17% | 25% |
| MS SQL | 29% | 19% | 21% | 27% |

Таблиця 5.12 - Використання оперативної пам’яті при виконанні пошуку числа більше заданого значення

|  |  |
| --- | --- |
| БД | Значення параметру |
| Cassandra | 346 Мб |
| MongoDB | 704 - 706 Мб |
| Redis | 550 - 1000 МБ |
| OrientDb | 488 Мб |
| MS SQL | 107 Мб |

У зазначеному розділ розглянуто порівняння баз даних згідно поставленої задачі.

ВИСНОВКИ

У даній роботі біло порівняно основні метода збереження даних у реляційних та нереляційних БД. Було проведено повний огляд їх структура, функціонування, методів роботи. Також було проведено порівняльний аналіз кількісних характеристик вибраних представників кожного типу СУБД. У результаті проведення експериментальних досліджень було вибрано СУБД, яка має кращі показники у вибраному критерії порівняння.

Серед кількісних характеристик було порівняно швидкість виконання запиту, кількість виділеної оперативної пам’яті та процесорне навантаження, яка виникає при виконанні певних запитів СУБД.

Немає єдиного рішення для всіх завдань. Найбільш близько до такого, незважаючи ні на що, - sql рішення. Кожне nosql сховище - це інструмент, який вирішує тільки певне коло завдань, при цьому вимагає роботи напилком, вивчення нутрощів і ретельну настройку, а то і написання свого клієнта.

Серед найшвидших СУБД було відібрано документо-орієнтоване сховище MongoDB та сховище типу “Key-Value” Redis. Але в той же час, ці два типи сховища виявилися такими, що потребують найбільшої кількості оперативної пам’яті.

Тому необхідно також звертатися до рішення реляційних баз даних з можливими оптимізуючими компонентами (індексами, додатковим шаром кешування і таке інше). Також реляційні СУБД проявляють себе як більш надійніші у якості відновлення даних після збоїв, підтримки транзакції оновлення даних та також мають дуже великий спектр можливостей, якими сьогоднішні NoSQL-рішення не володіють.

Популярність NoSQL-рішень зростає, і пов'язано це в першу чергу з новими завданнями з обробки що раніше не мислимих обсягів даних, з новими вимогами щодо доступності та масштабованості. Незважаючи на те що багато NoSQL-системи здаються (або пропагуються в такій якості) «срібною кулею», насправді це такі ж суперечливі рішення, як і традиційні реляційні СУБД. Адже, як відомо, не буває поганих інструментів - бувають інструменти, які підходять або не підходять під конкретні завдання.

ПЕРЕЛІК ПОСИЛАНЬ

1. Бібліотека розробника MSDN [Електронний ресурс] / портал [msdn.microsoft.com](http://msdn.microsoft.com) : Microsoft Developer Network. - Режим доступу: www/URL: <http://msdn.microsoft.com/library/>
2. Бондарєв В. «Об'єктно-орієнтовне програмування на C#»: навч.посіб. Харків, 2009. 224 с.
3. Крєнке Д. «Теорія та практика побудови баз данных. 8-е вид.»: СПб.: Питер, 2003. 800 с.
4. Ліберті Д. Програмування на C# Москва.: Россійське Видання, 2003. 688 с.
5. Лоусон, Б. Изучаем HTML5 второе издание Питер.: «Питер», 2012. 301с.
6. Мак-Дональд, М. Microsoft ASP.NET 3.5 з прикладами на C#, 2008.
7. Ріхтер, Дж. «CLR via C# Програмування на платформі Microsoft .Net Framework 4.0 на мові програмування С#», СПб.: Пітер, 2012. 929 с.
8. Робінсон С., Лорі М., Корнес О., Глінн Д. «C# для профессионалов», 2005. 396 с.
9. Сандерсон С. «ASP.NET MVC Framework с примерами на C# для профессионалов» Москів: ООО «И.Д. Вильямс», 2010. 560 с.
10. Сеппа Д. «Microsoft ADO.NET» СПб.: Пітер, 2003. 640 с.
11. Stack Overflow - Where Developers Learn, Share, & Build Careers. Stack Overflow. URL: <http://stackoverflow.com> (date of access: 23.11.2023)
12. Троелсен Е. «C# та платформа.Net»: СПб.: Пітер, 2006. 795 с.
13. MongoDB: The Developer Data Platform. MongoDB. URL: <http://www.mongodb.org> (date of access: 23.11.2023).

ДОДАТОК А

**ПРОГРАМНИЙ КОД**

Агрегація (знаходження середнього або максимального значення) у MS SQL Server

public float Aggregate(AggregateFunction function)

{

var res = 0.0f;

using (var connection = new SqlConnection(connectionString))

{

connection.Open();

var command = new SqlCommand

{

Connection = connection,

CommandText = string.Format("select {0}(Amount) from Requests", function)

};

res = (float) command.ExecuteScalar();

connection.Close();

}

return res;

}

start = DateTime.Now;

float avg = relatDbRepository.Aggregate(AggregateFunction.AVG);

stop = DateTime.Now;

Console.WriteLine("Average value {0} was found during {1} ms", avg, stop - start);

start = DateTime.Now;

float max = relatDbRepository.Aggregate(AggregateFunction.MAX);

stop = DateTime.Now;

Console.WriteLine("Max value {0} was found during {1} ms", max, stop - start);

Доступ до елементу за ідентифікатором

public Request FindById(object id)

{

Request request = null;

using (var connection = new SqlConnection(connectionString))

{

const string commandText = "select \* from Request where IdRequest = @idRequest";

var command = new SqlCommand { Connection = connection, CommandText = commandText };

command.Parameters.AddWithValue("@idRequest", id);

connection.Open();

var reader = command.ExecuteReader();

if (reader.Read())

{

request = new Request

{

IdRequest = reader.GetGuid(0),

BusinessType = reader.GetString(1),

Name = reader.GetString(2),

City = reader.GetString(3),

Country = reader.GetString(4),

Description = reader.GetString(5),

Phone = reader.GetInt64(6),

BusinessName = reader.GetString(7),

PostalCode = reader.GetString(8),

Address = reader.GetString(9),

Amount = reader.GetFloat(10),

UserId = reader.GetString(11),

IdStatus = reader.GetInt32(12),

Notes = reader.GetString(13)

};

}

connection.Close();

}

return request;

}

start = DateTime.Now;

var request = relatDbRepository.FindById(Guid.NewGuid());

stop = DateTime.Now;

Console.WriteLine("Find by Id {0}", stop - start);

Видалення об’єкту

public Request Remove(Request obj)

{

using (var connection = new SqlConnection(connectionString))

{

const string commandText = "delete from Requests where IdRequest = @idRequest";

var command = new SqlCommand {Connection = connection, CommandText = commandText};

command.Parameters.AddWithValue("@idRequest", obj.IdRequest);

connection.Open();

command.ExecuteNonQuery();

connection.Close();

}

return obj;

}

start = DateTime.Now;

relatDbRepository.Remove(request);

stop= DateTime.Now;

Пошук об’єкту по цілому значенню більше заданого параметру

public IList<Request> FindGreaterThan(long searchPhone)

{

var lists = new List<Request>();

using (var connection = new SqlConnection(connectionString))

{

const string commandText = "select \* from Request where Phone > @phone";

var command = new SqlCommand { Connection = connection, CommandText = commandText };

command.Parameters.AddWithValue("@phone", searchPhone);

connection.Open();

var reader = command.ExecuteReader();

while (reader.Read())

{

lists.Add(new Request

{

IdRequest = reader.GetGuid(0),

BusinessType = reader.GetString(1),

Name = reader.GetString(2),

City = reader.GetString(3),

Country = reader.GetString(4),

Description = reader.GetString(5),

Phone = reader.GetInt64(6),

BusinessName = reader.GetString(7),

PostalCode = reader.GetString(8),

Address = reader.GetString(9),

Amount = reader.GetFloat(10),

UserId = reader.GetString(11),

IdStatus = reader.GetInt32(12),

Notes = reader.GetString(13)

});

}

connection.Close();

}

return lists;

}

start = DateTime.Now;

requests = relatDbRepository.FindGreaterThan(45);

stop = DateTime.Now;

Console.WriteLine("Find by integer value greater than {0}", stop - start);

Пошук об’єкту по дробовому значенню менше заданого параметру

public IList<Request> FindLessThan(float searchAmount)

{

var lists = new List<Request>();

using (var connection = new SqlConnection(connectionString))

{

const string commandText = "select \* from Request where Amount < @amount";

var command = new SqlCommand { Connection = connection, CommandText = commandText };

command.Parameters.AddWithValue("@amount", searchAmount);

connection.Open();

var reader = command.ExecuteReader();

while (reader.Read())

{

lists.Add(new Request

{

IdRequest = reader.GetGuid(0),

BusinessType = reader.GetString(1),

Name = reader.GetString(2),

City = reader.GetString(3),

Country = reader.GetString(4),

Description = reader.GetString(5),

Phone = reader.GetInt64(6),

BusinessName = reader.GetString(7),

PostalCode = reader.GetString(8),

Address = reader.GetString(9),

Amount = reader.GetFloat(10),

UserId = reader.GetString(11),

IdStatus = reader.GetInt32(12),

Notes = reader.GetString(13)

});

}

connection.Close();

}

return lists;

}

start = DateTime.Now;

requests = relatDbRepository.FindLessThan(33.39f);

stop = DateTime.Now;

Console.WriteLine("Find by float value less than {0}", stop - start);

Пошук текстового значення (знаходження підстроки)

public IList<Request> FindByText(string strFind)

{

var lists = new List<Request>();

using (var connection = new SqlConnection(connectionString))

{

const string commandText = "select \* from Request where Description" +

" like '%'+@text+'%'";

var command = new SqlCommand

{

Connection = connection,

CommandText = commandText

};

command.Parameters.AddWithValue("@text", strFind);

connection.Open();

var reader = command.ExecuteReader();

while (reader.Read())

{

lists.Add(new Request

{

IdRequest = reader.GetGuid(0),

BusinessType = reader.GetString(1),

Name = reader.GetString(2),

City = reader.GetString(3),

Country = reader.GetString(4),

Description = reader.GetString(5),

Phone = reader.GetInt64(6),

BusinessName = reader.GetString(7),

PostalCode = reader.GetString(8),

Address = reader.GetString(9),

Amount = reader.GetFloat(10),

UserId = reader.GetString(11),

IdStatus = reader.GetInt32(12),

Notes = reader.GetString(13)

});

}

connection.Close();

}

return lists;

}

start = DateTime.Now;

requests = relatDbRepository.FindByText("Beer 2.0 L");

stop = DateTime.Now;

Console.WriteLine("Find by text {0}", stop - start);

ДОДАТОК Б

**ПРЕЗЕНТАЦІЯ**

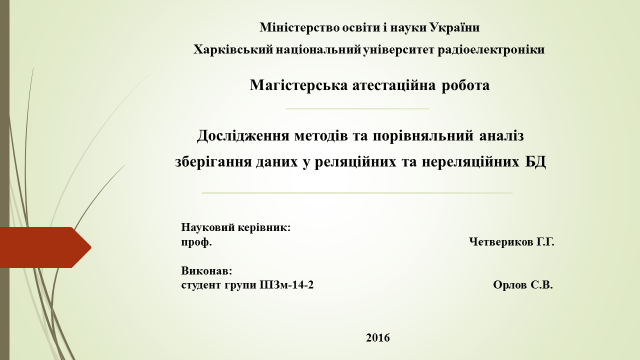


Рисунок Б.1

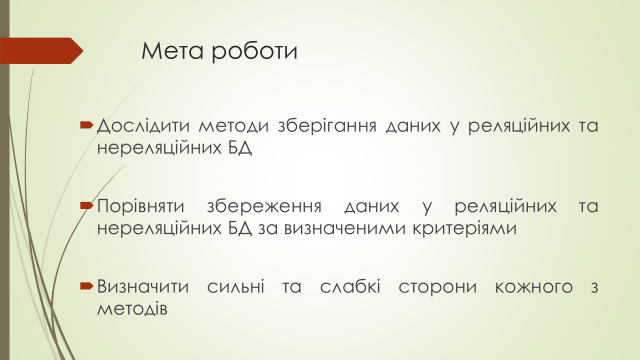


Рисунок Б.2

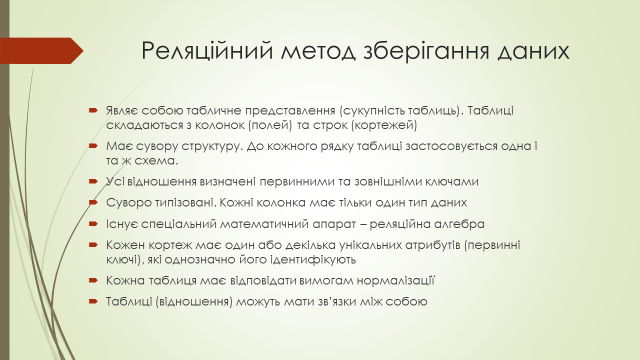


Рисунок Б.3

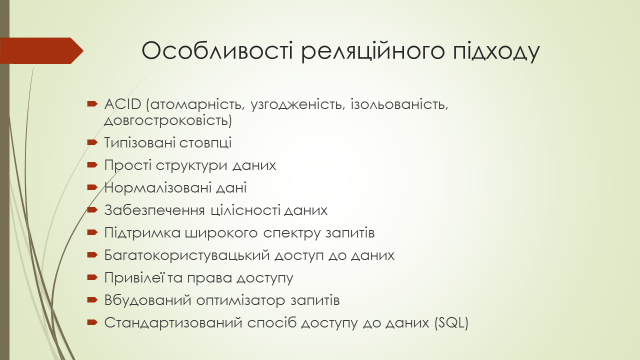


Рисунок Б.4

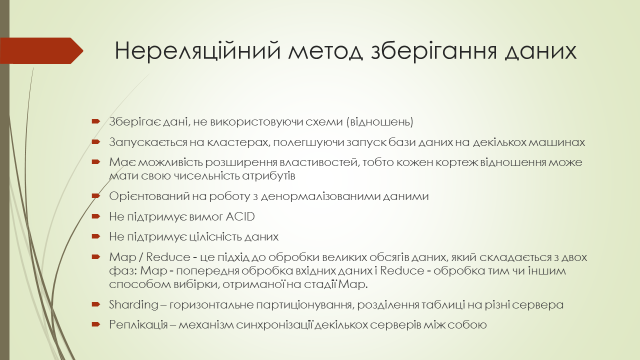


Рисунок Б.5

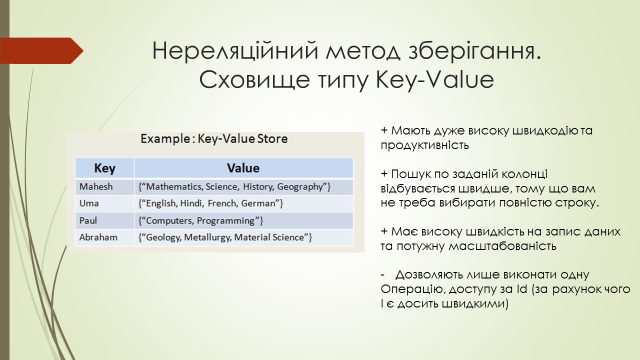


Рисунок Б.6



Рисунок Б.7

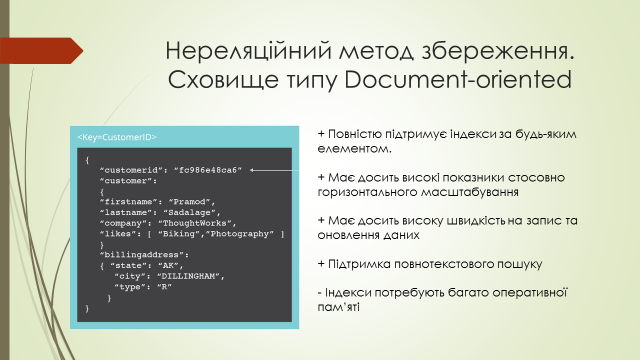


Рисунок Б.8



Рисунок Б.9

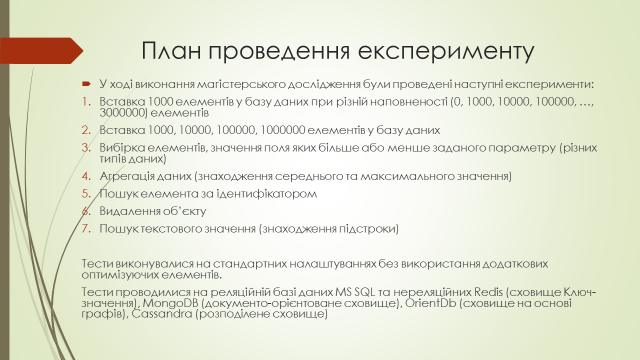


Рисунок Б.10

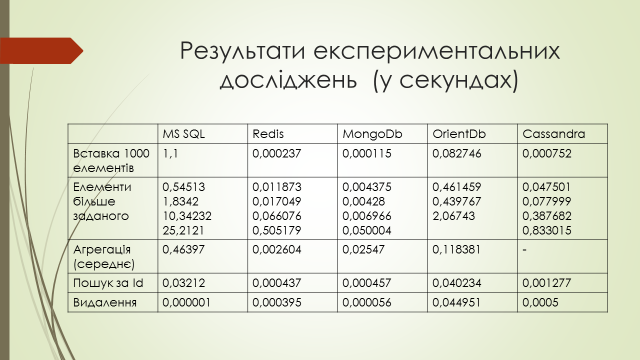


Рисунок Б.11

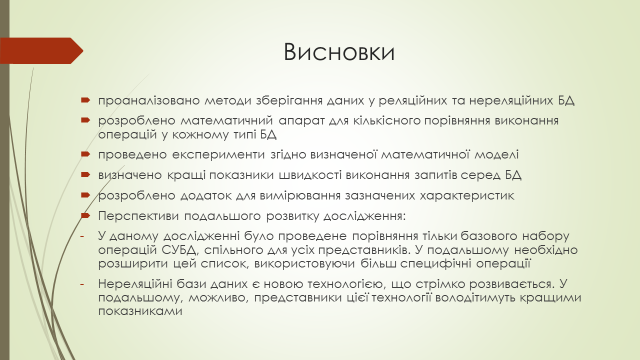


Рисунок Б.12

ДОДАТОК В

ТЕЗИСИ ДО НАУКОВОЇ КОНФЕРЕНЦІЇ

**СРАВНИТЕЛЬНЫЙ АНАЛИЗ**

**РЕЛЯЦИОННЫХ И НЕРЕЛЯЦИОННЫХ БД**

This work is devoted to the description of the main features of approach of data storage with using of not relational NoSQL databases comparing with SQL databases. In this work the main advantages of using NoSQL databases and the main shortcomings in comparison with use of traditional SQL databases are described. Also examined applications of different types databases and their advantages and disadvantages in comparison with traditional databases. In addition, main features of NoSQL and SQL databases were described and compared using mathematical model (different comparative coefficients were used for this purpose).

В данной работе рассматриваются не реляционные базы данных в качестве альтернативы существующим реляционным базам данных.

Целью исследования является сравнительный анализ реляционных и не реляционных БД с точки зрения производительности, масштабируемости, сложности обслуживания, безопасности, физического хранения, количества потребляемой памяти, структурированности, а также скоростных характеристик.

Объектом исследования является задача определения критериев выбора конкретного типа БД (не реляционного или реляционного) в зависимости от типа решаемой задачи. Следует определить преимущества и недостатки каждого подхода и проверить их практическим путем.

Предметом исследования является оценка и сравнительный анализ скоростных характеристик выполнения запросов, характеристик расходования памяти и структурных характеристик (изменения схемы базы данных) реляционных и не реляционных баз данных.

Постановка задачи: для выполнения заданной цели необходимо решить следующие задачи:

1. Сравнить скоростные характеристики запросов на стандарте SQL с запросами специфических стандартов JSON (NoSQL).
2. Произвести анализ на надежность обновления данных, используя транзакции в SQL и любые возможные средства в NoSQL.
3. Сравнить объем потребляемой памяти, а также относительную загрузку процессора при выполнении операций обращения к базе данных.
4. Произвести нагрузочное тестирование при использовании соответствующей БД на одном из серверов и сравнить полученные результаты.
5. Исследовать надежность, безопасность, структурированность при изменении/удалении данных по внешнему ключу или другому ссылочному элементу в не реляционных БД.

Решение задачи скоростных характеристик можно проиллюстрировать следующими рисунками (Рисунок 1).

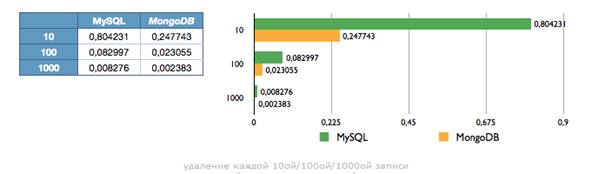


Рисунок 1 – сравнение скорости выполнения CRUD

В то время, как среднюю процессорную загрузку и потребляемую память по запросам можно представить таблицей:

|  |  |  |
| --- | --- | --- |
| NoSQL | 30%-40% | 704 — 1000 МБ |
| SQL | 5%-25% | 550 — 706 МБ |

Обобщенную математическую модель по основным показателям можно представить следующей формулой (обобщенный критерий оценки J):

J = F(V, C, M, R),

где V – скорость выполнения запроса, C – процессорная нагрузка, M – объем потребляемой памяти при выполнении запроса, R – коэффициент надежности (вероятность восстановление после сбоев).

Не реляционные базы данных предстали в качестве альтернативной модели реляционным базам, чтобы решить их основные проблемы (например, производительность). Но решив проблемы реляционных БД, они пожертвовали некоторыми характеристиками (например, надежностью). В результате, полностью отказаться от какого-либо из подходов (NoSQL или SQL) не удается, поэтому следует исходить из требований задачи.

1. J. Chris Anderson, Jan Lehnardt, Noah Slater. “CouchDB: The Definitive Guide”. O'Reilly Media, 2010.

2. Rick Cattel. “Scalable SQL and NoSQL data stores. ACM SIGMOD Record”. New York, NY, USA, December 2010.

3. Н. А. Скворцов. Отображение моделей данных NoSQL в объектные спецификации, 2012.

4. *Kalinichenko L.A., Stupnikov S.A., Martynov D.O.* SYNTHESIS: a Language for CanonicalInformation Modeling and Mediator Definition for

Problem Solving in Heterogeneous Information Resource Environments. – Moscow: IPI RAN, 2007. – 171 p.