Міністерство освіти і науки України

Національний університет «Львівська Політехніка»

Кафедра ЕОМ



**Магістерська кваліфікаційна робота**

Виконав:

ст.гр. СПРм-21

Приймак Д.В.

Керівник:

Акимишин О.І.

Львів – 2014

# АНОТАЦІЯ

Під час виконання даної МКР проведено дослідження існуючих підходів до побудови сховищ даних. Як показано в роботі, на протязі останніх 30 років такі програмні рішення розробляються з використанням СКБД та виділених потужних серверних станцій.

В дослідженні проведено аналіз переваг та недоліків таких програмних рішень. Також в ході виконання МКР було дослідження можливості побудови схожих програмних рішень в розподілених програмних системах на базі технології Hadoop. Було запропоновано спосіб побудови доволі гнучких конфігурованих виміро-орієнтованих сховищ даних на базі розподіленої системи. Як показано в роботі – така альтернатива дозволяє забезпечувати набагато кращу масштабованість системи, і в свою чергу виконувати основне завдання системи не зважаючи на суттєве збільшення обємів даних, які обробляються.

Результатом є прототип конфігурованого виміро-орієнтованого сховища даних на базі технології Hadoop, та його детальний опис. Подано також структурні схеми та алгоритм роботи системи.

# ABSTRACT

While working on current MKR there has been performed an investigation about existing ways of building Dimensional Data Warehouses. As it was mentioned in this work, such solutions are developed using the RDBMS and powerful server station already for almost 30 years.

There is an analysis of pros and cons of such program solutions. Also as a part of this MKR, there has been made a study about the possibility to build such king of solution based on distributed computational system Hadoop. The way of building flexible and configurable Dimensional Data Warehouse, based on such distributed platform, was proposed in this work. As it was shown – such an alternative makes it possible to provide better scalability of the system and as consequence to continue processing data even in case of it significant increasing.

The result of the MKR is a prototype of a configurable Dimensional Data Warehouse based on Hadoop technology stack and it’s precise description. Structural diagrams and schema describing work of algorithms are also provided.

# ЗМІСТ

[АНОТАЦІЯ 2](#_Toc405409044)

[ABSTRACT 3](#_Toc405409045)

[ЗМІСТ 4](#_Toc405409046)

[ВСТУП 6](#_Toc405409047)

[1. АНАЛІТИЧНИЙ ОГЛЯД 9](#_Toc405409048)

[1.1 Означення понятя сховища даних 9](#_Toc405409049)

[1.2 Огляд найбільш поширених підходів до організації архітектури сховищ даних 10](#_Toc405409050)

[1.3 Аналіз виміро-орієнтованого підходу в моделюванні даних 13](#_Toc405409051)

[1.4 Огляд основних етапів при побудові сховищ даних 17](#_Toc405409052)

[1.5 Огляд програмного забезпечення для побудови сховища даних 22](#_Toc405409053)

[1.5.1 База даних та апаратна частина 23](#_Toc405409054)

[1.5.2 Інструменти перетворення даних 24](#_Toc405409055)

[Висновок 26](#_Toc405409056)

[2. ОБГРУНТУВАННЯ ВИБРАНОГО НАПРЯМУ РОБОТИ 28](#_Toc405409057)

[2.1 Дослідження вимог до сховищ даних 28](#_Toc405409058)

[2.2 Дослідження шляхів побудови сховищ даних на основі платформи Hadoop 33](#_Toc405409059)

[2.3 Схема процесу перетворення вхідної інформації та підготовки сховища даних 47](#_Toc405409060)

[Висновок 51](#_Toc405409061)

[3. ПРОЕКТНИЙ РОЗДІЛ 54](#_Toc405409062)

[3.1 Проектування сховища даних 54](#_Toc405409063)

[3.2 Вибір технологій для побудови системи 55](#_Toc405409064)

[3.3 Розробка алгоритму формування таблиць вимірів 63](#_Toc405409065)

[Висновок 70](#_Toc405409066)

[4. ЕКСПЕРИМЕНТАЛЬНИЙ РОЗДІЛ 72](#_Toc405409067)

[Висновок 76](#_Toc405409068)

[5. ЕКОНОМІЧНИЙ РОЗДІЛі 78](#_Toc405409069)

[5.1 Розрахунок собівартості 78](#_Toc405409070)

[5.2 Розрахунок договірної ціни та прибутку НДР 82](#_Toc405409071)

[5.3 Оцінка наукової та науково-технічної результативності НДР 83](#_Toc405409072)

[Висновок 86](#_Toc405409073)

[ВИСНОВКИ 87](#_Toc405409074)

[СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 88](#_Toc405409075)

[ДОДАТКИ 90](#_Toc405409076)

# ВСТУП

Сховища даних є дуже поширеним типом систем у найрізноманітніших предметних галузях. Починаючи від торгівлі та страхування і завершуючи медициною та військовою галуззю. У всіх цих випадках важливим питанням є зберігання надзвичайно великих обємів даних у формі, придатній до швидкого та якісного аналізу. Дані повинні зберігатися у різних формах на протязі тривалого часу. В залежності від предметної галузі вимоги до часу доступу можуть коливатися від десятих частин секунди до кількох хвилин.

Найчастіше такі системи використовуються для побудови звітів та аналізу збережених даних. На базі цього аналізу виконуються прогнозування ситуації в предметній галузі та прийняття стратегічних рішень. Так в залежності від інформації, яка буде отримана з даних про продажі в мережі магазинів, залежатиме політика та стратегія розвитку цієї мережі. Також з даних про захворювання в тому чи іншому регіоні можна зробити корисні висновки для запобігання епідеміям та неконтрольваним спалахам тих чи інших захворювань.

Всі ци системи потребують високої продуктивності та надійності, адже від їх роботи залежить дуже багато процесів. Через невиконання цих вимог до системи компанії-користувачі щорічно втрачають шалені суми грошей.

Останні кілька десятиліть ця проблема розвязувалася з допомогою систем, побудованих на базі систем керування базами даних (СКБД) і інструментів для обробки та перетворення даних. Всі ці програмні продукти працювали на великих, потужних але дуже дорогих серверних машинах. Концептуальна модель сховищ даних досить сильно відрізняється від систем обробки транзакцій, які також досить поширені в різних предметних галузях. Основною відмінністю сховищ даних (які є прикладом OLAP, Online Analytic Processing, систем) від систем транзакційної обробки, є спосіб зберігання даних. Якщо в транзакційних системах дані зазвичай зберігаються у вигляді нормалізованих відношень, то в сховищах такі дані є денормалізованими і приведеними до вигляду, зручного для побудови звітів. Недоліком транзакційних систем є те, що при побудові складних звітів по великих обємах даних, необхідно виконувати операції перетину між кількома таблицями. А якщо такі таблиці мають значні розміри, то памяті в системі просто не вистачає для проведення таких перетинів за прийнятний час.

Основним способом, яким вирішується дана проблема, є попередня денормалізація даних і зведення їх до такого вигляду, який не потребує зєднань і перетинів між кількома таблицями. Це дозволяє не задіювати такі ресурси памяті, які необхідні в транзакційних системах.

Найпоширенішим шляхом реалізації сховищ даних є набори з СКБД, інструментів для перетворення даних, інструментів для інтеграції з різними джерелами та споживачами даних та системи для побудови звітів і проведення аналітичних досліджень. Такі підходи є хорошим рішенням при порівняно невисокій кількості даних, що мають зберігатися. Та у випадку стрімкого росту обємів вхідних даних, такі системи не надають достатньої масштабованості та гнучкості в розширені.

Щоб обійти таку проблему потрібно спроектувати та розробити таку систему, яку буде нескладно розширювати і змінювати з ростом величини вхідних даних.

Для цього потрібно виконати заміну в першу чергу способу зберігання даних – СКБД забезпечують тільки вертикальну масштабованість, а також – способу обробки даних – існуючі системи не розраховані на виконання в розподілених системах та кластерах з багатьох машин.

Хорошою альтернативою може бути побудова сховища даних на базі платформи розподілених обчислень MapReduce та її конкретної реалізації – Hadoop. Ця платформа забезпечує можливість написання розподілених програмних систем, які будуть працювати на кластерах з сотень або й тисяч звичайних машин. Дана платформа була розроблена в компанії Google і вже досить тривалий час використовується в системах з надвеликими обсягами даних, що обробляються.

В даний момент Hadoop це багатофункціональна розподілена файлова система, що забезпечує високу доступність та надійність зберігання файлів поверх якої працює бібліотека з високооптимізованою реалізацією парадигми програмування MapReduce. Система володіє майже горизонтальною масштабованістю і надає можливість інтеграції з великою кількість програмних систем.

Побудова сховища даних на базі такої платформи може забезпечити вирішення багатьох проблем, з якими стикаються розробники в даний момент. Враховуючи, що всі важливі частини по забезпеченню паралелізації, розподілу файлів між машинами та відмовостійкості інкапсульовані у вже готову реалізацію, фреймворк Hadoop, розробка такої системи повинна бути нескладною з точки зору розробників.

В даній магістерській кваліфікаційній роботі розглядається можливість та пропонується підхід до побудови виміро-орієнтованого сховища даних на базі розподіленої програмної платформи Hadoop.

1. **АНАЛІТИЧНИЙ ОГЛЯД**
   1. **Означення понятя сховища даних**

Різні люди по різному дають означення поняттю сховища даних. Найбільш популярне визначення належить Білу Інману(Bill Inmon), який дає таке означення:

**Сховище даних** – це предметно орієнтований, інтегрований, постійний і незмінний набір даних, призначений для прийняття управлінських рішень.

В даному визначенні варто розяснити деякі терміни:

**Предметно-орієнтований**: сховище даних може використовуватися для аналізу певної предметної області. Наприклад «торгівля» може бути предметною областю.

**Інтегрований**:сховище даних обєднує та уніфікує дані, що приходять з різних джерел. Наприклад джерело 1 та джерело 2 мають різні способи представлення однієї і тієї ж сутності. Але в сховищі даних ці види представлення будуть зведені до одного, спільного.

**Постійний:** Історичні дані зберігаються в сховищі даних. Наприклад є можливим отримати дані, що надійшли 3, 6, 9 місяців назад, або іншої довільної давності. Це досить сильно відрізняється від транзакційних систем, в яких зазвичай доступна тільки найновіша версія даних. Наприклад транзакційна система може утримувати тільки останню(поточну) адресу користувача, в той час, коли в сховищі даних зберігається інформація і про всі попередні адреси цього користувача.

**Незмінний:** як тільки дані попали в сховище даних, вони більше ніколи не піддаються змінам. Тому історичні дані в сховищах даних ніколи не повинні змінюватися.

Також досить відомим є таке визначення сховища даних, дане Ральфом Кімбалом(Ralph Kimball)

**Сховище даних** – це копія транзакційних даних, які спеціально структуровані для виконання запитів на них та аналізу.

Це є більш функціональний погляд на сховище даних. Кімбал не акцентує уваги на тому яким чином таке сховище побуловане(як це робить Інмон), а в свою чергу фокусує увагу на тому, який функціонал воно надає.

## 1.2 Огляд найбільш поширених підходів до організації архітектури сховищ даних

Різні сховища даних мають різну структуру. Деякі можуть мати операційне сховище даних, в той час як інші – декілька спеціалізованих сховищ (так званих кіосків або вітрин даних). Одні можуть мати невелику кількість джерел даних, а інші, в свою чергу – десятки таких джерел. Зважаючи на це, набагато кращим рішенням є представити архітектуру у вигляді набору різних шарів(рівнів) сховища даних, аніж аналізувати деталі конкретних систем.

В загальному випадку всі сховища даних маються такі рівні:

* Рівень джерела даних
* Рівень вибірки даних
* Зона проміжного збереження даних
* Рівень перетворення даних
* Рівень збереження даних
* Рівень логіки перетворень даних
* Рівень презентації даних
* Рівень метаданих
* Операційно-системний рівень

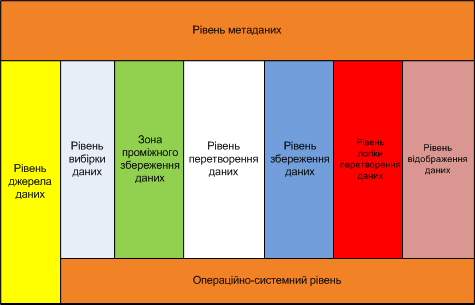


Рис. 1.1. Архітектура типового сховища даних

Надалі детальніше розглядається кожний з згаданих рівнів

**Рівень джерела даних**

Це рівень відображає різні джерела даних, з яких наповнюється сховище даних. Джерела даних можуть бути найрізноманітніших форматів – звичайні текстові файли, реляційні бази даних, інжі види баз даних, файли Excel і т.д.

Багато різних типів даних можуть бути джерелом даних. Наприклад:

* логи веб сервірів з даними про перегляд сторінок користувачами;
* дані внутрішніх ринкових досліджень;
* статистичні дані з державних установ;

Всі ці дані формують рівень джерела даних.

**Рівень вибірки даних**

Це рівень який відповідає за витягування даних з джерел і переміщення їх в систему сховища даних. На цьому етапі може відбуватися деяке очищення і перевірка даних, але без якихось суттєвих перетворень.

**Зона проміжного збереження даних**

Це зона, куди дані завантажуться перед початком подальших перетворень їх для сховища даних. Наявність однієї спільної зони для всіх отриманих даних робить легшим процес їх послідовної та повторної обробки.

**Рівень перетворення даних**

Це частина системи, яка відповідає за перетворення даних з зони проміжного зберігання за певними правилами та приведення до певної структури, придатної для подальшої аналітичної обробки.

**Рівень збереження даних**

Це місце де зберігаються перетворені та очищені дані. В залежності від обсягу та функціональності виділяють такі три сутності: сховище даних, вітрина даних та сховище операційних даних. В довільній системі може бути присутня як одна, так і всі види сутностей.

**Рівень логіки перетворення даних**

Це частина системи не зберігаються всі правила функціонування. Ці правила не впливають на схему перетворення, за якою готується сховище, але вони впливають на звіти, які генеруються на базі збережних даних.

**Рівень відображеня даних**

Це відноситься до інформації, яку безпосередньо отримує користувач системи. Під користувачем систем мається на увазі людина, яка займається отриманням цінності з даних, які зберігаються в сховижі. Це може бути наприклад бізнес-аналітик, який проводить аналіз рівня покупок в певній мережі магазинів, за якийсь період часу. Дані можуть відображатися у вигляді таблиць, графіків, діаграм. Звіти можуть генеруватися з будь-якою періодичністю та надсилатися електронною поштою. Зазвичай спеціальні інструменти для побудови аналітичних звітів використовуються на цьому рівні.

**Рівень метаданих**

Це рівень на якому зберігається інформація про дані, що знаходяться в сховищі даних. Логічна модель даних(схема даних) може бути зразком того, що зберігається на рівні метаданих.

**Операційно-системний рівень**

На цьому рівні зберігається інформація про функціонування сховища даних. Тут наприклад можуть зберігатися лог-файли виконання перетворення даних, звіти про продуктивність системи і т.д.

**1.3 Аналіз виміро-орієнтованого підходу в моделюванні даних**

Найчастіше в системах сховища даних використовується виміро-орієнтована модель даних. Вона відрізняється від третьої нормальної форми, що часто використовується в системах транзакційного типу(OLTP). З цього випливає, що дані в такій системі повинні зберігатися по-іншому.

Для того щоб описати виміро-орієнтовану модель даних варто спочатку означити основні часто вживані терміни.

**Вимір:** це категорія інформації. Наприклад часовий вимір.

**Атрибут:** це унікальний рівень в межах одного виміру. Наприклад, Місяць є атрибутом часового виміру.

**Ієрархія:** це специфікація рівнів, що відображає відношення між різними атрибутами в межах одного виміру. Наприклад одною з можливих ієрархій в часовому вимірі є Рік→Квартал→Місяць→День.

**Таблиця фактів:** таблиця фактів це таблиця, що містить заміри певного типу. Для прикладу, кількість продаж може бути таким заміром. Ці заміри зберігаються в таблиці фактів з певною гранулярністю. Для прикладу це може бути кількість продаж в одному магазині за один день. В цьому випадку таблиця фактів міститиме три колонки: дата, магазин, кількість продаж.

**Таблиця пошуку:** таблиця пошуку надає детальну інформацію про атрибути. Наприклад, таблиця пошуку для атрибуту Квартал буде містити список всіх кварталів, що наявні в сховищі даних. Кожний рядок(кожний квартал) може мати кілька полів: одне для унікального ідентифікатора ID, що ідентифікує квартал, і одне або більше додаткових полів, що специфікують як конкретний квартал відображений в звіті (наприклад перший квартал 2001 року може бути представлений як «Кв1 2001» або «2001 Кв1»)

Виміро-орієнтовна модель включає таблиці фактів та таблиці пошуку. Таблиці фактів зєднані з одною або кількома таблицями пошуку, але різні таблиці фактів не мають звязків між собою. Виміри та ієрархії представлені таблицями пошуку. Атрибути є колонками в таблицях пошуку, але не ключами.

В проектуванні моделі даних для сховищ даних або вітрин даних найбільш широко використовуються схеми двох типів: зіркоподібна схема та схема «сніжинка».

В залежності від того, яку бізнес-задачу потрібно розвязати, використовується одна з цих двох схем.

**Зіркоподібна схема**

При проектуванні сховища даних за схемою «зірка», єдиний обєкт(таблиця фактів) розміщується в центрі і радіально зєднується з оточуючими таблицям (таблицями пошуку) наче зірка. Кожний вимір представлений одною таблицею пошуку. Первинний ключ в таблиці пошуку звязаний з зовнішнім ключем в таблиці фактів.

Зразок схеми сховища виду «зірка»



Рис. 1.2. Зіркоподібна схема сховища даних

Всі заміри з таблиці фактів взаємоповязуються зі всіма вимірами(таблицями пошуку), з якими звязана таблиця фактів. Коротше кажучи вони мають однаковий рівень гранулярності.

Зіркоподібна схема може бути як простою так і складною. Проста схема складається тільки з однієї таблиці фактів. Коплексні можуть мати більше ніж одну таблицю фактів.

Для наглядності можна навести приклад. Припустимо, що наше сховище даних зберігає інформацію про продажі і різними вимірами є час, магазин, продукт та клієнт. В цьому випадку схема з рис. 2 показує як будуть звязані таблиці в нашому сховищі. Лінії відображають наявність відношення «первинний ключ – зовнішній ключ» між таблицями. Варто звернути увагу що різні виміри(таблиці пошуку) не звязані між собою.

**Схема «сніжинка»**

Схема «сніжинка» є розширенням зіркоподібної схеми. В такій схемі кожна точка зірки(вимір) розчеплюється на декілька точок. В схемі «зірка» кожний вимір представлений єдиною таблицею пошуку, в той час як в схемі «сніжинка» вимір предствлений нормалізованим набором з кількох таблиць пошуку, відображаючи таким чином різні рівні ієрархій.



Рис. 1.3. Схема «сніжинка»

Наприклад вимір часу складається з двох різних ієрархій

1. Рік→Місяць→День
2. Тиждень→День

В такому випадку ми матимемо 4 таблиці пошуку: одна для року, одна для місяця, одна для тижня та одна для дня.

Рік є звязаним з Місяцем, який звязаний з Днем. Тиждень звязаний тільки з Днем. На рис. 3 показано приклад схеми «сніжинка» з ієрархією для часового виміру.

Основною перевагою схеми «сніжинка» є покращення часу виконання запитів до сховища даних у звязку з мінімізацією використовуваного дискового простору та зєднування менших таблиць. Основним недоліком такої схеми є те, що доводиться нести додаткові витрати на підтримання всіх таблиць. А це стає доволі затратно, при рості кількості таблиць пошуку.

## 1.4 Огляд основних етапів при побудові сховищ даних

**Гранулярність таблиці фактів**

Першим кроком при проектуванні таблиці фактів є визначення гранулярності цієї таблиці. Під гранулярністю розуміється найдрібніший рівень інформації, що зберігатиметься в таблиці фактів. Для виконання цього пункту слід виконати такі кроки:

1. Визначити які виміри будуть включені в сховище даних.
2. Визначити де в ієрархії таблиць пошуку зберігатиметься кожна окрема інформація.

Ці кроки зазвичай здійснюються в залежності від бізнес вимог конкретної прикладної задачі.

**Визначення вимірів для включення в сховище даних**

Визначення вимірів, що будуть включені в сховище даних зазвичай є доволі простим процесом, адже такий перелік залежить від задачі, яка розвязується з допомогою сховища даних.

Для прикладу, в класичних системах, призначених для зберігання даних про продажі товарів, вимірами можуть бути бути Час, Географія та Продукт. Також можуть бути випадки, коли в систему включаються й додаткові виміри. Наприклад в певних мережах магазинів, для конкретних клієнтів можуть існувати знижки, або інші правила обслуговування, в залежності від всіх попередніх покупок цього клієнта. Для забезпечення такого функціоналу потрібно мати можливість розглядати збережену інформацію і з виміру Клієнт. Це означає, що в таке сховище даних потрібно буде включити також таблицю пошуку зі всіма клієнтами системи.

**Визначення до якої таблиці в ієрархії відноситимуться дані**

Визначення в якій частині ієрархії таблиць пошуку зберігати ту чи іншу інформацію не є жорстко детермінованим процесом. Це є одне з тих місць, де вибір цілком залежить від бізнес-задачі яку виконуватиме система.

Так у прикладі з сховищем даних в сфері продажу багато залежитиме від того, наскільки часто створюватимуться звіти. Якщо звіти генеруватимуться щогодини, то є зміст вибрати «годину» як найменший рівень гранулярності в таблиці, що відображає вимір Час. Якщо ж, звіти генеруватимуться щодня, то цілком достатньо вибрати найменшою одиницею гранулярності «день». З цього випливає, що чим детальніший рівень гранулярності, тим більшою буде таблиця фактів.

Задача зводиться до визначення балансу між детальністю збережуваної інформації та дисковим простором.

**Типи фактів та таблиць фактів**

**Типи фактів**

Існує три види фактів:

* Адитивні: адитивними фактами є факти, що можуть бути підсумовані (агреговані) через кожен з вимірів в таблиці фактів.
* Напів-адитивні: напів-адитивні це факти, що можуть бути підсумовані тільки через частину вимірів з таблиці фактів, але не через усі.
* Неадитивні: неадитивні факти це такі факти, що не можуть бути підсумовані через жоден з вимірів.

Варто розглянути приклад, що ілюструє всі типи фактів. В контексті першого прикладу вважатимемо що ми роздрібні торгівці і в нас є таблиця фактів з такими колонками.

Табл. 1.1. Таблиця адитивних фактів

|  |
| --- |
| Дата |
| Магазин |
| Продукт |
| Обсяг\_продаж |

Основним призначенням цієї таблиці є зберігати інформацію про обсяги продажу певного продукту в певному магазині. Дані беруться на щоденній основі. **Обсяг продаж** є фактом. В цьому випадку **Обсяг продаж** є адитивним фактом, адже ми можемо підсумувати цей факт через будь-який з трьох вимірів, що присутні в таблиці фактів – дата, магазин та продукт. Наприклад ми можемо підсумувати Обсяг продаж за всі сім днів тижня і таким чином отримаємо підсумок обсяку продаж за повний конкретний тиждень.

Як другий приклад можна розглянути ситуацію, коли ми є банком і маємо таку таблицю фактів:

Табл. 1.2. Таблиця напів-адитивних та неадитивних фактів

|  |
| --- |
| Дата |
| Рахунок |
| Поточний\_баланс |
| Прибуток |

Призначенням цієї таблиці є запис та збереження балансу та прибутку для кожного рахунку в кінці кожного дня. Поточний Баланс та Прибуток є фактами. Поточний баланс є напів-адитивним, адже є зміст підсумувати це поле для всіх рахунків, щоб отримати поточний баланс банку, але немає змісту підсумовувати в Часовому вимірі (сумування всіх щоденних балансів для конкретного рахунку не дасть нам жодної корисної інформації). Прибуток є неадитивним фактом, адже нема змісту сумувати їх значення у вимірі рахунків або на щоденному рівні.

**Типи таблиць фактів**

Виходячи з класифікації фактів, розділяють два типи таблиць фактів:

* **Кумулятивна**: цей тип таблиці фактів відображає що відбувалося на протяз певного періоду часу. Для прикладу в такій таблиці можуть зберігатися всі продажі всіх продуктів по всіх магазинах за день. Таблиці цього типу зазвичай містять адитивні факти. Приклад таблиці, наведеної в описі адитивного типу фактів є кумулятивною таблицею фактів.
* **Зріз**: такий тип таблиць фактів відображає стан речей в певний момент часу і зазвичай включає здебільшого напів-адитивні та неадитивні факти. Таблиця з станом банківських рахунків, що наведена в попередньому прикладі є зразком таблиці зрізу.

**Повільно-змінювані виміри**

Проблема повільно-змінюваних вимірів є доволі поширеною в сфері сховищ даних. Суть проблеми полягає в тому, що є випадки, коли значення атрибуту для певного рядка змінюється з часом. Для кращої наглядності варто навести приклад:

Марія є клієнтом компанії. Спершу вона проживала в місті області А. І тому первинний запис в таблиці пошуку клієнтів сховища даних цієї компанії виглядав так:

Табл. 1.3. Приклад таблиці повільно-змінюваного виміру

|  |  |  |
| --- | --- | --- |
| Ключ\_клієнта | Імя | Область |
| 1001 | Марія | Область А |

Через деякий час вона переїхала в область Б. Отже як команія повинна зберігати таку зміну в сховищі даних? Потрібно враховувати що деякі звіти чи запити можуть потребувати як дані про теперішній так і про минули стан атрибуту.

В загальному випадку є три способи для вирішення такого виду проблем. В залежності від способу розвязання, виділяють такі три типи повільнозмінюваних вимірів.

Тип 1: новий запис ігнорується. Постійно зберігаються тільки первинні дані.

Тип 2 : новий запис повністю заміняє старий. Жодні історичні дані не зберігаються

Тип 3 : новий запис додається до таблиці пошуку з вказуванням дати, коли він став активним. Старий запис позначаєть застарілим і зберігає дати, для зазначення часового проміжку, коли він був актуальним.

## 1.5 Огляд програмного забезпечення для побудови сховища даних

Беручи до уваги те, що сховища даних є великими та складними системи, їх побудова потребує використання великої множини інструметів. Для побудови кожного з рівнів потрібний надійний та розвинутий програмний продукт.

При обранні програмних інструментів для побудови сховища даних варто звертати уваги на такі напрями:

* база даних та апаратна частина
* інструмент перетворення даних
* інструмент здійснення он-лайн аналітичної обробки
* інструмент для побудови звітів
* спосіб зберігання метаданих про сховище даних

Варто розглянути основні вимоги до інструментів кожного з напрямків, а також уже існуючі програмні продукти.

### 1.5.1 База даних та апаратна частина

При обиранні цього типу інструментів варто звертати увагу на такі критерії оцінювання:

* масштабованість: основними питаннями, яких стосується цей критерій є «як може рости система зі збільшення потреб в зберіганні?», «яка СКБД та апаратна платформа може обробляти набори даних найбільш ефективно?». Для того щоб правильно дати відповідь на такі питання потрібно перш за все визначити які обєми даних очікується обробляти і вже виходячи з цього вибирати конкретну систему.
* можливість паралельної обробки: беручи до уваги, що більшість сучасних систем мають велику кількість процесорних ядер та центральних процесорів, варто враховувати це при обранні системи.
* сумісність апаратних засобів та СКБД: оскільки СКБД в кінцеовму результаті працюють на обраному апаратному забезпечення, то варто уникнути можливих несумістностей ще не етапі вибору. Цілком можливо, що певний функціонал СКБД не підтримується певною апаратною реалізацією, що в свою чергу спричинить помилки в системі.

Найбільш поширеними СКБД, що використовуються для побудови сховищ даних є:

* [Oracle](http://www.oracle.com/)
* [Microsoft SQL Server](http://www.microsoft.com/)
* [IBM DB2](http://www.ibm.com/)
* [Teradata](http://www.teradata.com/)
* [Sybase](http://www.sybase.com/)
* [MySQL](http://www.mysql.com/)

Популярними операційними системами, під управлінням яких будують сховища даних є:

* [Linux](http://www.linux.org/)
* [FreeBSD](http://www.freebsd.org/)
* [Microsoft](http://www.microsoft.com/)

### 1.5.2 Інструменти перетворення даних

Коли доходить справа до вибору інструментів для перетворення даних незавжди є доречно купувати якісь сторонні інструменти. Рішення чи купувати чи розробляти інструмент самому залежить від таких трьох чинникі:

* складність трансформацій даних: чим складнішими є перетворенням тим більш раціонально буде придбати сторонню програму для перетворень
* потреба в очищенні даних: чи потребують дані очищення перед тим як піддаватись перетворенням і перед збереженням їх в сховищі даних. Якщо так, то варто обрати інструмент з потужним функціоналом для очищення даних. В інших випадках доцільніше розробити таку програму самому
* розмір даних. Наявні комерційні продукти зазвичай мають в наявності доповнення, що здатні пришвидшити рух даних. Тому при потребі обробити значні обєми даних доцільніше буде обрати якийсь сторонній інструмент.

**Функціональні можливості інструментів для перетворення даних**

Якщо вибір бази даних та апаратної платформи є абсолютною необхідністю, то вибір інструменту для обробки даних не настільки важливим питанням. При обранні інструментів для перетворення даних варто звертати увагу на такі аспекти:

* функціональні можливості: тут маються на увазі як і частину, що забезпечує трансформації даних, так і частину, що відповідає за очищення даних. Зазвичай трапляються інструменти, що володіють потужною функціональністю в питанні перетворень даних, або в їх очистці і доволі рідко трапляються продукти, що володіють цими функціональними властивостями в однаково повній мірі. Якщо нам відомо про велику ймовірність того, що вхідні дані будуть неакуратними, то варто звернути увагу на якісний процес очищення даних. І в той же час, якщо відомо що дані потребуватимуть складних перетворень, то варто віддати перевагу інструменту з хорошим функціоналом перетворення.
* Можливість читати прямо з джерела даних: для кожної організації джерела даних можуть відрізнятися. При виборі інструменту для перетворень варто пересвідчитись що він може прямо зєднуватись з наявним джерелом даних.
* Підтримка метаданих: інструмент для перетворення даних грає ключову роль у ваших метданих адже він проводить відображення вхідних даних на вихідні, що є дуже важливою частиною метаданих. В деяких організаціях покладаються на документацію до інструментів як на основне джерело метаданих. Як результат – варто ретельно вибирати інструмент з врахуванням стратегії взаємодії з метаданими.

Найпоширенішими інструментами для перетворення даних є

* IBM WebSphere Information Integration (Ascential DataStage)
* Ab Initio
* Informatica
* Talend

# Висновок

Сховища даних є важливим типом систем, що широко використовуються в багатьох предметних галузях. За останні 25 років в цій сфері переважали рішення на основі СКБД та великих серверних машин. Увесь цей час запропонованих рішень вистачало для опрацювання існуючих обємів даних. Та на протязі останніх 5-7 років кількість даних, що потрібно було зберігати почала зростати за експоненціальним законом. Існуючі рішення не володіють достатньою масштабованістю та можливістю справлятися з даними такого обєму в припустимі відрізки часу. На протязі всього існування сховищ даних масштабування відбувалося тільки вертикально – за рахунок встановлення потужніших серверних машин, на яких сховища даних працювали. Також продуктивність таких систем покращувалася шляхом програмних оптимізацій та удосконалень у самій концептуальній структурі системи.

Але все частіше такі способи масштабування стають неприпустимими через висок вартість, складність реалізації або просто недостатній ефект. Це означає, що повинна бути запропована альтернатива існуючим підходам. Вона повинна надати можливість легко і порівняно недорого збільшувати продуктивність системи, дозволяючи таким чином обробляти обєми даних, що ростуть все стрімкіше.

Концептуальна модель сховищ даних може бути досить легко перенесена на нову розподілену програмну платформу Hadoop. Ця платформа дозволяє запускати досить складні обчислювальні процеси на великих кластерах зі звичайних машин. Концепція, що покладена в основу цієї платформи забезпечує майже лінійне горизонтальне масштабування. Сховища даних на основі цієї платформи є досить перспективним рішенням в плані побудов високонадійних, високонагружених масштабованих систем оброки та зберігання даних.

Нове рішення повинно забезпечувати весь функціонал, який до цього часу реалізовувався з використанням СКБД та інструментів перетворення даних. Також таке рішення повинно передбачати тісну і легку інтеграцію з існуючими джерелами та споживачами даних. Беручи до уваги бізнес-задачі які покладаються на системи сховищ даних, альтернативне рішення повинно забезпечувати високу відмовстійкість та доступність системи.

# 2. ОБГРУНТУВАННЯ ВИБРАНОГО НАПРЯМУ РОБОТИ

Перш ніж почати проектування та розробку системи необхідно чітко визначити які вимоги така система повинна виконувати. Тільки після повного розуміння та означення вимог програмне рішення може бути правильно спроектованим, реалізованим та відлагодженим. Типові сховища даних можуть забезпечувати не тільки зберігання великих обємів даних в придатному для аналітичних запитів вигляді, але й дозволяти обробку нових даних, що надходять, в реальному часі. Такі сховища даних можуть використовуватися для надання статистичної інформації про певну предметну область з врахуванням нових даних, що постійно надходять. В залежності від того, які вимоги ставляться до сховища даних, його структура та перелік використаних технологій значною мірою відрізнятиметься.

## 2.1 Дослідження вимог до сховищ даних

В загальному випадку розрізняють два основних види сховищ даних:

1. сховища даних що забезпечують виконання аналітичних запитів до великих обємів даних;
2. сховища даних які забезпечують окрім виконання аналітичних запитів ще й обробку в реальному часі нових вхідних даних.

Перший вид сховищ використовується в більшій мірі в наукових та дослідницьких організаціях. Також значного поширення вони набули в організаціях, які оперують великими обємами даних для побудови прогнозів та прийняття рішень на основі цих прогнозів. Характерною рисою таких систем є те, що вимоги до часу відповіді (латентності) не є критичними. В залежності від обсягів даних та використаної інфраструктури час відповіді таких систем може вимірюватися десятками хвилин або годинами.

Характерною особливістю другого виду систем є забезпечення можливості обробки великих обсягів нових вхідних даних з (або без) використанням даних, що вже збережені в сховищі. До таких програмних рішень ставляться досить жорсткі вимоги по часу відклику. Вони найчастіше використовуються в організаціях, які мають справу з даними, що швидко змінюються в часі і потребують максимально швидкого реагування.

Схему руху даних даних в системі першого виду можна зобразити так, як показано на рис 2.1.

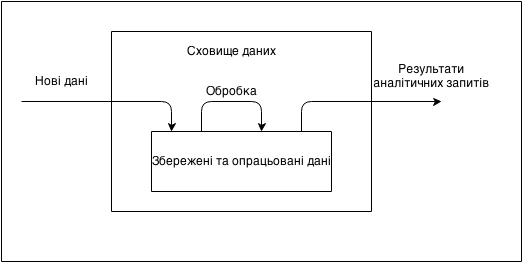


Рис. 2.1. Схема руху даних в сховищах для аналітичних запитів

Як видно зі схеми, в таких системах процес отримання нових даних є незалежним по відношенню до споживання даних зі сховища. Аналітичні запити до сховища даних використовують збережені та опрацьовані в системі дані. Дані можуть надходити в такий вид сховищ з найрізноманітніших джерел: вивантажуватись з систем керування базами даних, отримуватись у вигляді текстових файлів, у спеціальних текстових форматах і т.п. Основною вимогою до таких систем є забезпечити можливість зберігання та подальшого використання завантажених в систему даних. Час на опрацювання вхідних даних та приведення їх до вигляду, придатного для аналітичних запитів може займати суттєвий час. Основною причиною цього можуть бути:

- великі обсяги даних;

- складність трансформацій, що мають бути виконані над вхідними даними;

- необхідність використання всіх, раніше збережених даних для трансформації нових (в такому випадку час опрацювання зростатиме з накопиченням нових записів).

Споживачами даних, збережених у таких сховищах, зазвичай є аналітичні системи, які виконують велику кількість складних запитів до значних частин збереженої інформації. Обсяги інформації, в таких системах можуть сягати кількох десятків або й навіть сотень терабайт. Великою перевагою для проектувальників та розробників таких систем є те, що не ставляться жорсткі вимоги до часу опрацювання та підготовки даних. В першу чергу це дає можливість знехтувати певними оптимізаційними питаннями, які могли б відобразитися у вигляді значних часових затрат на розробку системи. Також це надає можливість зменшити вимоги до апаратної частини системи.

Сховища такого виду є поширеними уже на протязі тривалого часу і зазвичай будуються з використанням БД та великих серверних станцій. Використання таких технологій цілком достатньо і раціонально у випадку, якщо обєми вхідної і оброблюваної інформації не перевищують кількасот гігабайт. З ростом даних такі системи стають надто повільними, а програмних оптимізацій стає недостатньо щоб задовольнити вимоги. Єдиним шляхом є покращення апаратного забезпечення, а це надто дорогий крок.

Системи другого виду забезпечують не тільки можливість виконання аналітичних запитів, але й обробку та видачу вхідної інформації у реальному часі. Такі системи здатні реагувати на події, що виникають, та реагувати на них з надзвичайно низькою латентністю. Сховища такого виду зазвичай мають кілька інтеграційних точок: інтерфейс для виконання аналітичних запитів, та інструмент для передачі результатів обробки вхідної інформації, що безпереревно надходить в систему.

Схема руху даних, які потребують обробки в реальному часі зображена на рис. 2.2.



Рис. 2.2. Схема руху даних в сховищах з підтримкою відповіді в реальному часі

Цей вид сховищ даних є розширеним варіантом першого виду. Такі системи забезпечують можливість не тільки виконання аналітичних запитів, але й надають інтерфейс для виконання обробки вхідних даних в реальному часі.

Інформація в сховище надходить кількома різними шляхами. Це можуть бути великі завантаження файлів з файлових систем або таблиць з баз даних. Ці дані зберігаються та опрацьовуються для подальшого використання інструментами побудови аналітичних звітів. Окрім цього існує також канал, по якому в систему надходить інформація, яка потребує дуже швидкої обробки і передачі вже опрацьованих даних користувачам системи. Такі дані можуть оброблятися за наборами складних правил а також потребувати доступ до уже наявних даних. Вимоги до латентності такої системи є критичними. На відміну від першого типу сховищ даних, в яких дозволений час для опрацювання і перетворення даних може становити десятки хвилин або й години, в системах другого типу дані повинні опрацьовуватися за лічені секунди. В певних випадках час відклику має становити кількасот мілісекунд. Складністю побудови таких систем є в першу чергу те, що при значному збільшені вимог до часу відклику, вимоги до обсягів даних, які мають опрацьовуватися не зменшується. Це означає, що має бути досягнута надзвичайно велика паралелізація обчислень. Потік вхідних даних, які потребують обробки в реальному часі є безперервним, тому не допускаються жодні затримки або збої в системі. Швидкість такого потоку може досягати кількох Мб/с.

Сховища такого виду можуть використовуватися в організаціях, що надають оперативну інформцію в сферах, де потрібно опрацьовувати значні обєми даних в реальному часі. Наприклад метеостанції, медичні заклади, дослідницькі центри, які працюють з вимірюваннями швидкоплинних процесів.

Перелік вимог до систем значною мірою вплине на структуру програмного рішення. В магістерській кваліфікаційній роботі буде досліджено та розроблено шляхи побудови як першого так і другого виду сховищ даних. Для першого типу також буде запропонована реалізація, на основі платформи розподілених обчислень Hadoop.

## 2.2 Дослідження шляхів побудови сховищ даних на основі платформи Hadoop

Вимоги до системи є найбільш визначальним фактором, що впливає на весь процес проектування та розробки програмного рішення. Так і у випадку з сховищами даних, від способу обробки даних залежитиме набір технологій і спосіб їх взаємодії в межах системи. Платформа Hadoop надає можливість зберігання та паралельної обробки великих масивів даних на кластерах зі звичайного недорого обладнання. Такі кластери є набагато дешевшими аніж великі серверні станції, що використовуються для встановлення на них баз даних. Системи на основі Hadoop володіють майже лінійною масштабованістю, а це в свою чергу дозволяє легко і недорого розширювати апаратне забезпечення.

Для правильної побудови сховища даних потрібно враховувати те, як дані мають рухатись в системі. Під параметрами руху маються на увазі:

* кількість джерел даних
* інтерфейси, які ці джерела надають;
* обсяги даних;
* швидкість надходження;
* правила перетворень
* вимоги до часу зберігання
* інтерфейси систем-споживачів
* вимоги до часу запиту

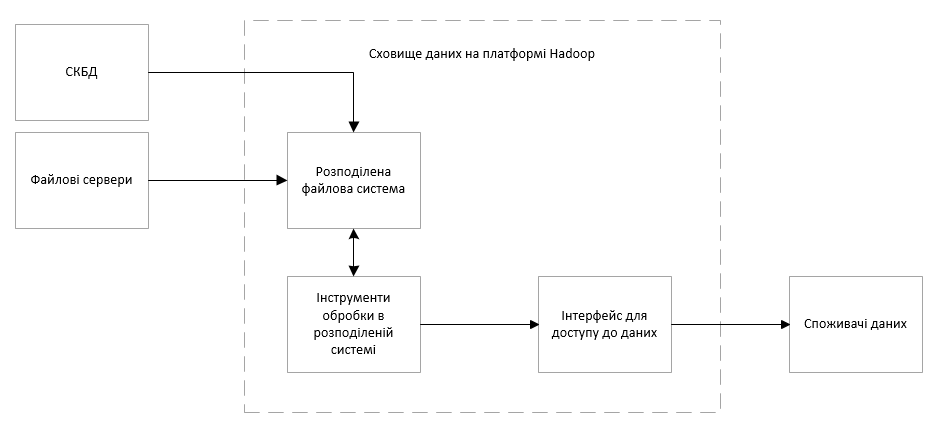
Всі ці параметри визначають спосіб взаємодії між усіма елементами системи та алгоритми їх роботи. Також від значень цих параметрів залежить спектр конкретних інструментів та технологій, які будуть використовуватися для побудови сховища даних.

Для початку розглянемо структуру програмного рішення сховища даних на основі платформи Hadoop.В звязку з тим, що сховища даних є предметно-орієнтованими, кількість та природа джерел даних буде різною в кожному окремому випадку. Але не зважаючи на це, існують певні аспекти, спільні для всіх систем такого типу. Перелік цих аспектів може повторно використовуватися в сховищах з різних прикладних сфер. Характеристики, якими системи відрізняються можуть бути винесені в окремий незалежний модуль. До таких характеристик можна віднести логічну модель даних, що зберігаються та правила обробки цих даних. Інтерфейси, інфраструктура та набір технологій можуть бути однаковими у найрізноманітніших предметних областях. Розділювання одної системи на такі незалежні модулі дозволить зробити її конфігурованою і забезпечити можливість повнорного використання її компонентів.

Спершу варто розглянути структуру сховища даних для виконання аналітичних запитів на ньому. Основними вимогами до такої системи є:

* інтерфейс для отримання даних з БД;
* інтерфейс для отримання текстових файлів з файлових серверів;
* можливість надійного зберігання надвеликих обємів даних;
* опрацювання та перетворення всіх даних за прийнятний час;
* інтерфейс для виконання аналітичних запитів на збережених опрацьованих даних;
* прийнятний час виконання запитів, що звертаються до всього обєму збережених даних.

Цей перелік вимог визначає структуру такої системи. Архітектура такого програмного рішення зображена на рис 2.3.

 Рис. 2.3. Структурна схема сховища даних для побудови аналітичних звітів на основі збережених даних

Далі буде детально розглянуто роль кожного елемента в системі та способи їх взаємодії. В наступному розділі цієї МКР для кожного елемента структурної схеми буде підібрано відповідну технологію і запропоновано більш детальну архітектурну діаграму.

Така система має два основних інтерфейси вхідних даних:

1. інтерфейс для роботи з СКБД;
2. інтерфейс для роботи з файловими серверами.

Наявність саме таких інтерфейсів зумовлена тим, що в даний момент майже у всіх предметних областях бази даних є найбільш використовуваним способом збереження даних. Тому при побудові нових програмних рішень варто враховувати, що великі обсяги даних мають бути перенесені з уже існуючих БД і можливо перетворені в інші логічні структури. Також важливим фактором є те, інформація, яка отримується від клієнтів та буде збережена в сховищі даних, в першу чергу заносить в бази даних. Це спричинено тим, що БД є досить оптимальним рішенням для швидкого збереження невеликої та середньої кількості даних. А сфери генерування інформації в яких безпосередньо задіяна людина характеризуються невеликою кількістю згенерованих даних. Потрібно також враховувати що існує велика кількість різних СКБД, тому інтерфейс сховища має забезпечувати сумісність з найпопулярнішим та найпоширенішими СКБД.

Важливим питанням є те, що СКБД хоч і забезпечують виконання великої кількості одночасних запитів, та можуть не справлятися з їх обробкою, якщо такі запити буде виконувати цілий кластер машин (а у випадку розподіленої системи це буде саме так). Тому інтерфейс повинен забезпечувати одночасні запити до СКБД-джерел тільки обмеженої кількості машин кластера. Особливо гострим таке питання може стати, якщо час вивантаження даних з БД буде співпадати з часом, коли БД забезпечує доступ до даних своїм першочерговим клієнтам (користувачам системи де основні дані продукуються). Це може призвести до відмови системи на достатньо тривалий час. А таке є недопустимим, адже може призвести до значних збитків.

Іншим не менш важливим інтерфейсом є інтерфейс з файловими серверами (наприклад FTP). В багатьох предметних областях основні відомості зберігаються в текстових файлах, де записи розділяються комами або іншими розділювачами. Файли з такими даними можуть досягати колосальних розмірів, а тому не можуть бути нормально опрацьовані на звичайних компютерах. Сучасні засоби обробки текстових файлів не дозволяються запускати опрацювання таких великих документів на кількох машинах і це ставить в безвихідне становище системи, яким така обробка необхідна. Файлові сервери можуть працювати за різними протоколами, тому інтерфейс сховища даних має враховувати це і надавати можливість працювати з найпоширенішими протоколами.

Після пересилання інформації з БД або файлових серверів вона має бути надійно збережена і готова для подальшої обробки. Для цього система повинна володіти надійною розподіленою файловою системою. Окрім надійності зберігання даних також має бути передбачений захист цих даних. Дуже часто дані, що мають бути опрацьовані, носять конфіденційний характер і вимагають надійного захисту від несанкціонованого доступу. Це можуть бути медична, фінансова та інша інформація.

Розподілені системи можуть містити десятки а то і сотні машин в свому складі. А це значно збільшує ймовірність відмови однієї або кількох машин кластера. Це має бути враховано при побудові розподіленої файлової системи. Адже при виході з ладу одного або кількох вузлів кластера можлива безповоротна втрата частини збережених даних. В більшості предметних областей це недопустимо. Існує кілька шляхів забезпечення надійності в розподілених файлових системах. Одним з найпоширеніших є реплікація даних. При такому підході вся інформація, що надходить в ФС реплікується і розсилається на кілька вузлів. Тому у випадку виходу з ладу однієї або кількох машин втрачені дані можуть бути відновлені з копій, що знаходилися на інших машинах. Недоліком таких систем є інтенсивне використання зовнішньої памяті. В залежності від степеня реплікацій, обсяги памяті, що займається може зрости в кілька разів по відношенню до розміру початкових даних. Також недоліком подібних систем є те, що потрібно вирішувати проблему несинхронізованості копій. Тобто у випадку редагування файлу, потрібно забезпечити щоб усі копії цього файлу були однаковими і містили останні зміни. Важливим елементом таких розподілених ФС є необхідність в наявності координатора та сховища метаданих. Їх призначенням є координування звертань до файлів у системі. Вони мають забезпечувати направлення звертання саме до того вузла, який містить необхідний файл. Координатор та сховище метаданих є найвразливішим місцем всієї файлової системи. Адже при їхньому виході з ладу втрачається можливість доступу до будь-якого з раніше збережених файлів.

Для забезпечення захисту збереженої інформації може використовуватися шифрування усіх файлів. Такий шлях захисту хоч і вберігає файли від несанкіонованого доступу, але значною мірою впливає на продуктивність системи. Шифрування та дешифрування при кожному звертанні до даних може в кілька разів зменшити швидкодію системи. Можливе також шифрування на апаратному рівні, але такий спосіб може мати дуже великий вплив на вартість обладнання. В такому випадку нівелюється можливість побудови розподіленого середовища з використанням звичайних машин.

Дані, які потрапили в розподілену систему повинні опрацьовуватися за певними правилами. Ці правила є унікальними для кожної предметної області і тісно повязані з природою даних. Так, у сховищах призначених для зберігання медичної інформації, вся інформація що надходить в систему може перетворюватися за складними правилами для того щоб бути представленою в необхідному вигляді. Необхідність такої обробки ставить досить високі вимоги до обчислювальних характеристик кластера. Система повинна забезпечити можливість паралельної обробки великих масивів даних. Зрозуміло, що такого неможливо досягти на звичайних однопроцесорних машинах. Для таких потреб необхідні великі розподілені системи. Відомо, що програми, які розроблені для звичайних обчислювальних машин, не можуть правильно виконуватися на розподілених системах, адже при їньому проектуванні не розглядалась можливість паралельної обробки кількох джерел даних, або великих кількостей вхідних даних. Для вирішення цієї проблеми була розроблена на реалізована програмно нова парадигма програмування MapReduce. На відміну від обєктно орієнтованого програмування, в такій парадигмі існує тільки невелика кількість основних обєктів. Це так звані мапери та редюсери (від Map – відображати, та Reduce – зменшувати). Велика кількість таких обєктів запускається як процеси на виконання на кожній з машин кластера. Вони виконують паралельні обчислення на даних, що надходять в систему, або ж уже зберігаються в ній. Детальніше цей підхід описаний у статті, яку в 2004 році випустили інженери компанії Google[посилання на статтю]. В даний момент існує велика кількість програмних реалізацій цієї парадигми. Така реалізація і покладена в основу платформи Hadoop.

На даний час існує кілька основних реалізацій. Принципова різниця між ними полягає в організації двох аспектів:

* шляхи керування ресурсами кластера;
* вид памяті, що використовується для зберігання проміжних результатів.

У раніших версіях платформи Hadoop використовувався централізований підхід до розподілу ресурсів. Розподілом ресурсів кластера (процесорний час, основна память, дисковий простір) займався один єдиний процес. В сучасніших версіях цієї платформи керування ресурсами є більш децентралізовано. На кожному з вузлів кластера встановлений менеджер ресурсів, які займається локальним керуванням. Але він узгоджує свої дії з центральним процесорм, що відповідає за координацію між усіма вузлами кластера.

Важливою різницею в способі реалізації є також вид памяті, що використовується для збереження проміжних даних обчислень. Якщо раніше такі результати зберігалися в зовнішній памяті кожного з вузлів розподіленої системи, то тепер все більше набуває поширення варіант реалізації, коли такі дані зберігаються в основній памяті. Це значною мірою збільшує швидкодію системи, але в той же час збільшує вимоги до апартних ресурсів кластера.

На рис 2.4 зображено схему руху даних в реалізації обсилювальної платформи зі збереженням проміжних даних в зовнішній памяті.

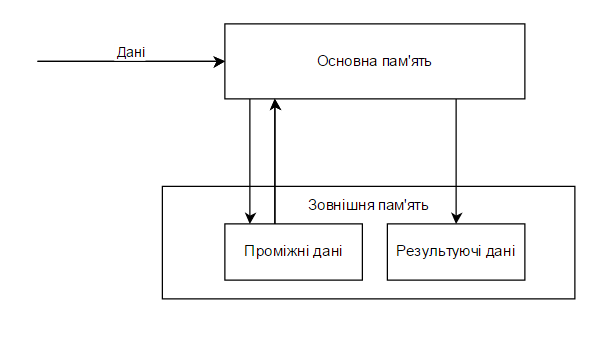


Рис. 2.4. Схема збереження даних в реалізації з використанням зовнішньої памяті для проміжних даних

Як можна бачити з цієї схеми, при такому шляху організації збереження даних виконується дуже велика кількість операції звернення до зовнішньої памяті. Такі операції є дуже часомісткими. Результатом цього є зниження швидкодії системи. Це обмеження можна обійти, якщо повністю перенести збереження проміжних даних в основну память. В такому випадку всі операції запису і читання, що необхідні при виконанні обчислень здійснюються до основної памяті. Час таких звернень є на порядок швидшим і тому це може суттєво збільшити загальну швидкість виконання обчислень. Схему руху даних при такому шляху організації памяті зображено на рис. 2.5.

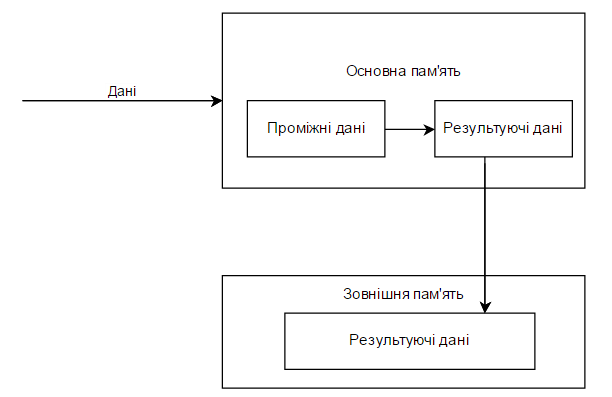


Рис. 2.5. Схема збереження даних в реалізації з використанням зовнішньої памяті для проміжних даних

Як видно зі схеми, при такому розподілі памяті зовнішній диск використовується тільки для збереження результатів обчислень. Кількість звернень до нього суттєво зменшилась в порівнянні з попереднім підходом, а тому й значною мірою збільшилася загальна швидкість обчислень. Реалізації з таким розподілом памяті набуваються все більшого поширення, адже вони дозволяють будувати системи з можливістю відклику в реальному часі. Результуючі дані при такому використанні памяті можуть надсилатись відразу споживачам даних, уникаючи таким чином аж двох часоємких кроків: збереження результату в зовнішню память та вичитування споживачем даних інформації з зовнішньої памяті.

Сховища даних також повинні володіти хорошим інтерфейсом для інтеграції з зовнішніми системами. Беручи до уваги, що зазавичай споживачами даних є аналітичні системи, які виконують запити до сховища та генерують аналітичні звіти на основі цих даних, досить важливою вимогою є забезпечення швидкого обміну інформацією між ними. Обсяги даних, які вичитуються з сховища є суттєвими і тому організація такого обміну вимагає додаткових зусиль. Найбільш ефективним інтерфейсом для взаємодії з сторонніми системами є SQL-інтерфейс до даних, які організовані у вигляді таблиць. Не завжди доцільним є формувати нормалізовані таблиці. Велика кількість зєднань між таблицями може призводити до значних вимог до основної памяті (в якій зберігатимуться результуючі таблиці). Досить поширеною практикою є денормалізація даних. При такому підході логічна структура даних відображається у «широкій таблиці» (таблиці з великою кількістю колонок). При такій організації ймовірною є неефективність зберігання інформації, адже однакові значення можуть повторюватися у великій кількості записів. Але виконувати запити до таикх таблиць є набагато менш ресурсомісткою задачою. Сховище даних повинно забезпечувати прийнятний час відклику при виконанні запитів до інформації.

Поширеним інтерфейсом з такими системами є також надання доступу до даних через веб-сервіси. Система може декларувати інтерфейс у вигляді RESTfull сервісів і таким чином забезпечувати можливість звертання до збереженої інформації у альтернативно до SQL вигляді. Реалізація інтерфейсів в найбільшій мірі залежить від особливостей систем-споживачі даних. Тому проектувати зовнішній інтерфейс потрібно тільки після аналізу вимог та дослідження вхідних інтерфейсів систем-споживачів.

Сховища даних, з підтримкою обробки вхідної інформації в режимі реального часу мають віддмінну від попередньо описаних, структуру системи та правила взаємодії між частинами цієї системи. Структурні модулі при такому підході повинні виконовувати іншу роль. Також значно відрізняються вимоги до часу відклику цих модулів. Якщо в системах для побудови аналітичних звітів основний акцент ставиться на можливість забезпечення складних перетворень надзвичайно великої кількості даних, а вимоги до часу відклику мають не такий великий пріоритет, то в системах з підтримкою обробки в реальному часі, питання швидкості обробки є критичним. Такі системи зазвичай виконують порівняно не складну обробку вхідних даних. Але забезпечують паралельну обробку великої кількості даних в дуже стислі часові терміни.

Структура такої системи та основні звязки між її модулями зображена на рис. 2.6.

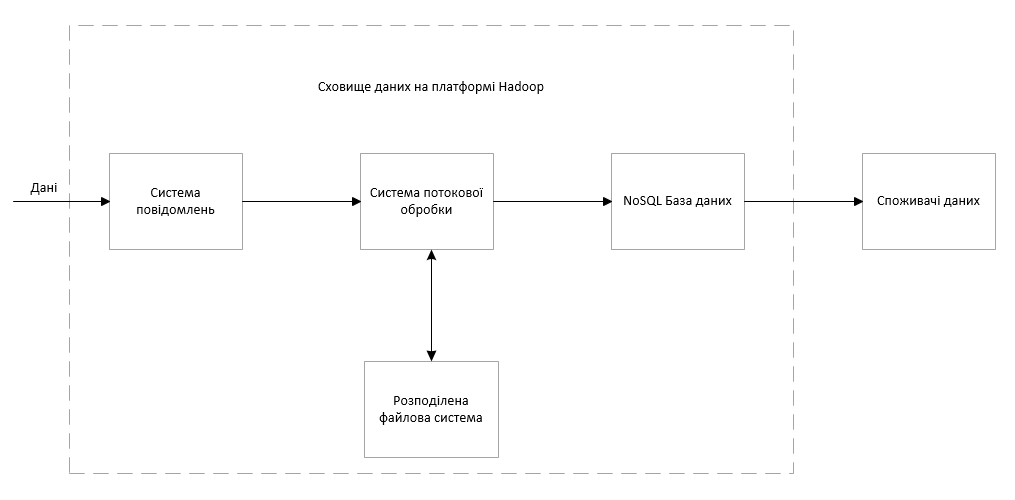


Рис. 2.6. Структурна схема сховища даних з можливістю обробки в реальному часі

Як видно з рис. 2.6. структура такого сховища даних суттєво відрізняється від структури, яку ми розглядали раніше.

Вхідним інтерфейсом для такого виду рішень слугує система повідомлень. Повідомленням може бути не тільки набір текстової інформації, але й будь-яка цілісна бінарна інформація. Одним з найважливіших призначень такої системи повідомлень є роль буфера вхідних даних у системі. У випадку, якщо б вхідні дані потрапляли одразу в систему потокової обробки, за певних обставин могла б скластись ситуація, що обчислювальний модуль не справляється зі всією кількістю вхідної інформації, тому деякі частини цієї інформації могли б бути втраченими. У багатьох прикладних сферах це є недопустимим. Для уникнення такої ситуації вводиться буфер вхідних даних у вигляді системи повідомлень. Найкращим рішенням є використовувати розподілену систему повідомлень, що в свою чергу забезпечує надійність та продуктивність цього модуля.

З системи повідомлень збережені дані вичитуються модулем для потокової обробки інформації. Цей модуль може реалізовуватися як розподілена обчислювальна система, яка виконує свої програми на великому кластері. У платформі Hadoop існують такі механізми. Вони забезпечують можливість паралельно обробляти великі кількості вхідних даних з надзвичайно малим часом відклику. Дуже часто для обробки вхідних даних потрібно звертатися до вже збереженої інформації. Такі правила звертання залежать від конкретної предметної сфери, але все ж не можна відкидати таку потенційну вимогу. Проблемою, що може виникати в такій ситуації є значна різниця у швидкості обробки та часу звертання до даних, що вже існують в системі. Операція звертання є набагато повільнішою, тому щоб час відклику не ставав неприйнятно великим потрібно виконувати певні оптимізації в організації таких звертань. Це може бути побудова індексів для збережених даних. Це дозволить значно пришвидшити довільний доступ, уникаючи необхідності виконувати повне сканування даних. Також хорошим рішенням є використання кешів, які знаходитимуться в памяті і містимуть дані, до яких найчастіше виконується звертання. У випадку деяких систем, велику кількість уже збереженої інформації можна зберігати безпосередньо в основній памяті і тільки періодично записувати її в зовнішню память та підвантажувати нові дані.

Характер даних, для яких необхідні системи обробки в реальному часі, дуже часто суттєво відрізняються від тих даних, що потрібні для генерування аналітичних звітів. Такі недавні не завжди легко перетворити у табличну структуру, що в свою чергу не дає можливості забезпечити для них SQL-інтерфейс. Також важливою вимогою є швидкість відклику при звертанні до зовнішнього інтерфейсу, тому в такій системі не працюватимуть шляхи, запропоновані раніше. Хорошим варіантом організації зовнішнього інтерфейсу подібних сховищ даних є NoSQL бази даних. Основною їх перевагою, в порівнянні з звичайними БД є можливість зберігати погано структуровані або неструктуровані дані. Великою перевагою таких БД є також можливість їх функціонування в розподілених системах. Час читання та запису в такі БД є порівняно низьким, тому вони є одним з найоптимальніших рішень для використання у ролі зовнішнього інтерфейсу.

Платформа Hadoop є достатньо розвинутою і включає в себе великий перелік програмних продуктів, що спрощують організацію складних розподілених обчислень. В тому числі до її складу входять кілька інструментів потокової обробки даних та NoSQL база даних, яка надає можливість запису та читання надзвичайно великих осягів даних майже у реальному часі. Всі ці елементи платформи добре між собою інтегруються. Це значно спрощує процес побудови програмних рішень, та зменшує кількість часу, необхідного на відлагодження.

## 2.3 Схема процесу перетворення вхідної інформації та підготовки сховища даних

Кількість та складність кроків при побудові кожного конкретного сховища даних може дуже відрізнятися. Ці параметри в першу чергу залежаться від предметної області, характеру даних та завдань, які потрібно виконувати з використанням цих даних. Але не зважаючи на таке різноманіття певні кроки в цьому процесі залишаються однаковими у всіх випадках.

У більшості сховищ даних перед тим як дані будуть готові для використання, над ними мають виконатися такі кроки:

* завантаження в систему;
* очищення від пошкоджених частин та непотрібної інформації;
* валідація;
* перетворення провалідованих даних за бізнес-правилами та підготовка таблиці фактів(вимірювань);
* генерація таблиць вимірів з провалідованих даних;
* підготовка опрацьованої інформації у вигляді, необхідному для споживачів.

Кожний з кроків в цій послідовності є важливим і має бути виконаним без помилок та від відхилень. При неможливості або неправильності виконання кожного з кроків, подальший запуск наступних не має змісту, адже отримана після таких дій інформація буде некоректною і непридатною для споживання.

На рис. 2.7. зображена схема послідовності виконання операцій над даними в системі.

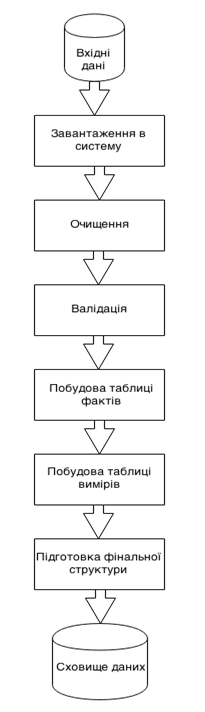


Рис. 2.7. Схема виконання основних операцій в процесі побудови сховища даних

Далі розглянемо кожен з кроків процесу побудови сховища даних детальніше. Не залежно від того, для якої предметної галуз будуються сховище даних, всі наведені кроки мають бути присутні. Відрізнятися вони можуть тільки бізнес-правилами за якими перетворюються дані. Саме ця частина логіки системи може бути винесена в окремий змінний модуль, яким і відрізнятимуться реалізації сховищ. Вся інша частина логіки діяльності системи є однаковою для різних реалізацій.

Для того, щоб дані можна було опрацьовувати їх потрібно спочатку отримати в систему. Це виконується під час кроку завантаження. На цьому етапі послідовності вся вхідна інформація завантажується і зберігається без змін. Важливо наголосити, що на цьому кроці ніяких перетворень з інформацією не відбувається. Це робиться для того, щоб якщо станеться збій на одному з наступних етапів, можна було повторно запустити процес з одними і тими ж даними. Також на вимогу користувачів можуть відбуватися повторні запуски підготовки результуючих таблиць з використанням тільки певної множини раніше отриманих даних (наприклад тих, що надійшли за перший місяць попереднього року). Така дія була б неможливою, якщо б не існувало копії вхідних даних без змін.

Наступним етапом в процесі є очищення даних. Під очищенням розуміються дії з відкидання пошкоджених файлів та видалення записів, обробка яких неможлива. Такими записами може бути якась службова інформація, що без необхідності змішана безпосередньо з даними. Для прикладу це може бути перелік назв колонок на початку на в кінці кожного файлу з записами. Ця інформація є важливою, але перетворень над нею робити не потрібно. Тому вона відкидається на цьому етапі.

Після очищення відбувається валідація даних. В залежності від специфіки інформації, що обробляється, на неї можуть накладатися певні обмеження. Для прикладу, якщо серед полів кожного з записів є поле, в якому зберігається інформація про вік людини, то ці значення мають бути в діапазоні від 1 до 130. Отже якщо буде отримано запис зі значення -250 в такому полі, він має вважатися недійсним і відкидатися. Також в процесі валідаці замість відкидання цілого запису неправильні значення можуть замінятися певними значеннями по замовчуванню.

Оскільки дуже поширеними є виміро-орієнтовані сховища даних, то в процесі обробки вхідної інформації повинні окремо формуватися таблиці вимірів та таблиці фактів. Зазвичай таблиці вимірів формуються шляхом витягування з вхідних даних значень певної групи полів, та запису їх в окремі таблиці за певними правилами. В залежності від того, який тип виміру був обраний для того чи іншого поля, ці таблиці формуватимуться за доволі складними схемами. Після завершення цього в кроку в системі буде збережено набір таблиць вимірів.

Наступним кроком є опрацювання всієї вхідної інформації та формування з неї таблиць фактів. При виконанні цих дій використовуються дані з попередньо сформованих таблиць вимірів. Правила побудови таких таблиць фактів, можуть бути надзвичайно складними і використовувати додаткову, зовнішню інформацію. Цей етап, зазвичай, є найбільш ресурсомістким і займає найбільше часу. По завершенні цього кроку в системі створюються або оновлюються уже існуючі таблиці фактів, а в них зберігається вся нова інформація.

Завершальним етапом в процесі формування сховища даних є надання доступу до опрацьованої та збереженої інформації у вигляді фінальної структури. Цей крок може виконуватися шляхом надання доступу у вигляді SQL інтерфейсу до збережених таблиць, або переносом даних з цих таблиць у якусь іншу підсистему.

Описаний вище процес може запускатися періодично. Тригером для такого запуску може слугувати поява нових даних. Такий спосіб не завжди є доцільним, адже ресурсоємкість всього процесу обробки є велика і запускати його заради кількох нових записів є неефективно. Частіше такі запуски роблять з певними часовими інтервалами, на протязі яких надходить достатньо велика кількість нових даних.

## Висновок

В цьому розділі було оглянуто основні вимоги які ставляться до сховищ даних та дослідженно два шляхи побудови таких сховищ. Кожний з запропонованих шляхів має свої плюси та мінуси і повинен обиратися в залежності до характеру вимог, та необхідної продуктивності системи.

Як перший так і другий шлях побудови використовує як основу розподілену платформу Hadoop. На її основі може бути реалізовано як виміро-орієнтоване сховище даних для побудови аналітичних звітів, так і система що дозволяє виконувати обробку великих обсягів інформації у реальному часі. Внутрішньо такі архітектури значною мірою відрізняються. Також відрізняються вимоги, які такі системи ставлять до апаратного забезпечення. Так, беручи до уваги, що в системах для аналітичних звітів не ставляться жорсткі вимоги до часу виконання, то для такої системи може бути використаний кластер з машин не таких потужний яку випадку системи, що підтримує обробку в реальному часі.

Беручи до увагу різницю в русі даних у системах, також відрізнятимуться вимоги до памяті кластерів. У випадку системи для обробки в реальному часі рекомендовано використовувати машини з основною памяттю великого обєму.

Як одна так і інша система є доволі складна для розробки. Досить вагомим чинником, що ускладнює процес розробки таких систем є велика складність функціонального тестування та відлагодження продукту. Оскільки системи є розподіленими, то правильно відтестувати їх можна тільки в розподіленому середовищі. Кожний запуск супроводжується великими часовими витратами. Характерною особливістю розподілених програмних продуктів є те, що існують певні накладні витрати часу при кожному запуску, для підготовки до виконання і запуску службових механізмів. Тому якими б малими не були обсяги даних для обробки, та є певний мінімум часу, який завжди потребуватиме кожний запуск програми.

Відлагодження таких програмних продуктів також є складним процесом. Треба враховувати що багато частинок роботи виконуються на множині машин. Розробник повинен мати це на увазі в процесі написання коду. А у випадках коли виникає необхідність відлагодити таку систему – розуміти і памятати всі механізми, за якими працює розподілена система. Хоч і наявні інструменти дають можливість максимально абстрагуватися від низькорівневих питань, та все ж повне розуміння деталей виконання може допомогти уникнути фундаментальних помилок в процесі розробки.

В наступному розділі буде запропоновано перелік технологій та розроблено архітектурну схему програмного рішення сховища даних, призначеного для виконання аналітичних запитів та побудови аналітичних звітів. Схема буде доповнена детальним описом процесу розробки, конфігурування та розгортання системи. Побудова система також включає в себе розгортання віртуального кластера та установки і конфігурування на ньому платформи Hadoop. Цей процес має бути максимально автоматизованим і довзоляти виконати всі необхідні дії за максимально короткий відрізок часу.

# 3. ПРОЕКТНИЙ РОЗДІЛ

В попередньому розділі було досліджено кілька шляхів побудови сховищ даних на основі платформи Hadoop. Результатом цього стало коротке порівняння двох наведених шляхів та структурні схеми обох наведених рішень.

В даному розділі буде спроектовано та детально описано список всіх компонентів системи та перелік технологій, обраних для реалізації. Потрібно брати до уваги факт, що технології в світі Hadoop розвиваються дуже стрімко і вже за короткий час весь перелік інструментів, обраних в даній роботі, може бути застарілим. Але навіть не зважаючи на це, таке спроектовано рішення дозволить підбирати та оновлювати всі частини системи шляхом заміни компонентів на більш нові. Платформа спроектована так, що інтеграція та оновлення компонентів системи є доволі легким і не протребує багато роботи розробника.

Далі структурна схема, розроблена в попередньому розділі, буде розширена деталями про логіку роботи та руху даних. Така схема зможе слугувати вихідною точкою для безпосередньої розробки програмного рішення і виконувати роль карти на шляху побудови системи.

* 1. **Проектування сховища даних**

В попередньому розділі було розроблено структурну схему сховища даних, призначеного для виконання аналітичних запитів на побудови аналітичних звітів. Було наведено призначення кожної структурної частини системи та способои інтеграції частин між собою. Тепер мною буде запропоновано низькорівневий опис роботи системи та перелік технологій для побудови такого сховища даних.

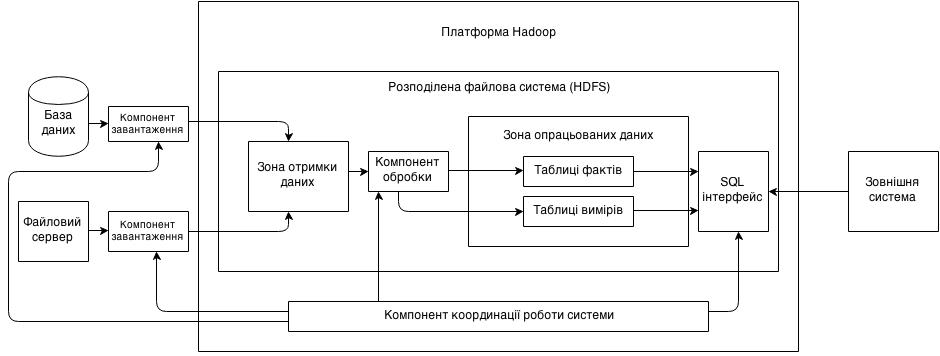


Рис. 3.1. Схема компонентів системи

В ході проектування мною буде обрано конкретні технології та інструменти для реалізації схеми наведеної на рис. 3.1. Як можна побачити, ця схема більш детально описує структуру системи, на відміну від схеми архітектури, наведеної в попередньому розділі. Основною різницею є наявність компонентів завантаження, обробки та координації роботи системи. Також в цій схемі детальніше наведено розмежування даних в розподіленій файловій системі. Таке розділення відображає логічні зони для даних, що існуватимуть в системі. В зоні отримання даних вся інформація зберігатиметься в такому вигляді, як вона надійшла з баз даних та файлових серверів. Згідно до структури побудови виміро-орієнтованих сховищ даних, всі дані будуть розбиті на таблиці фактів та таблиці вимірів, які і формуватимуть схему «зірка» або ж схему «сніжинка».

* 1. **Вибір технологій для побудови системи**

Оскільки система будуватиметься на основі платформи Hadoop, то одним з основих технологічних елементів, які потрібно обрати є дистрибутив Hadoop. Дистрибутивом платформи вважається пакет інтегрованих між собою та правильно налаштованих елементів екосистеми Hadoop. Сам по собі Hadoop це просто фреймворк для розподілених обчислень та розподілена файлова система. Всі елементи можна встановлювати самому, в ручному режимі, але це дуже ресурсоємкий процес, який потребує глибинних знань платформи. В даний момент є три найпоширеніших постачальника дистрибуцій: Hortonworks, Cloudera та MapR. З трьох описаних варіантів мною було вибрано постачальника Hortonworks. Ця компанія пропонує зовсім безплатно дистрибуцію HDP(Hortonworks Data Platform) 2.2. Цей набір включає в себе найновіші версії всіх основних компонентів екосистеми. Основними елементами, що входять до складу HDP 2.2 є:

* Hadoop & YARN(Yet another resource negotiator) 2.6.0
* Pig 0.14.0
* Hive & HCatalog 0.14.0
* HBase 0.98.4
* Spark 1.2.0
* Kafka 0.8.1
* Sqoop 1.4.5
* Ambari 1.7.0
* Oozie 4.1.0

Як видно зі списку – перелік технлогій доволі великий (а це тільки невелика частина), тому налаштувати їх вручну було б складно. Більшість наведених інстументів буде використовуватися при побудові системи. Далі буде наведено короткий опис призначення цих інструментів та спосіб, в який вони будуть використані при побудові сховища даних.

Основними компонентами, на яких власне й базуються всі обчислення в платформі є фреймворк Hadoop там менеджер ресурсів YARN. Це в першу чергу розподілена файлова система HDFS(Hadoop Distributed File System). Вона забезпечує надіє на ефективне зберігання великої кількості файлів на кластерах з великої кількості машин. Алгоритми розподілу даних в такій системі забезпечують велику швидкість доступу, читання та запису файлів. Файлова система також забезпечує високу найдійність за рахунок реплікакації даних. Менеджером ресурсів в платформі є компонент YARN. Він відповідає за розподіл виконання завдань по кластері з машин, та координує процес запуску, роботи, та збору результатів. Як HDFS так і YARN це набори процесів master та slave, які працюють на кожній з машин кластеру та спілкують з використанням мережних протоколів. Master-процеси виконують надсилання завданнь slave-процесам а також слідкують за їх виконанням. Всі обчислення, що відбувають в платформі Hadoop виконуються за посередництвом цих двох складових. Вони, на ряду з Map та Reduce завданнями є найбільш низькорівневими та фундаментальними блоками всіх Hadoop-систем.

Платформа Hadoop написана на мові Java та виконується як набір процесів у віртуальній машині Java. Це означає, що на цій мові можна написати будь яку програму, якщо в ній дотримана парадигма MapReduce, яка буде запускатися на кластері. Недоліком такого підходу є те, що як тільки логіка програми ставатиме порівняно тяжкою, буде надзвичайно важко відобразити її в термінах конструкцій мови Java. Для уникнення цих труднощів було розроблено кілька скриптових мов, які дозволяють набагато легше маніпулювати даними та виконувати їх перетворення. Однією з таких мов є скриптова мова Pig. В ній основними термінами є табличні структури даних та операції над ними. З допомогою цеї мови можна дуже легко будувати конвеєри обробки даних. Так з допомогої її конструкцій можна описати перелік трансформацій, які потрібно виконати над задекларованим набором даних, вказати джерело даних, місце та спосіб збереження та запустити на виконання. Транслятор цієї мови перетворить весь написаний код у набір Java MapReduce завдань, які будуть запущені на виконання в системі Hadoop. Використання такої мови значно пришвидшує і полегшує процес розробки. Особливо зручно використовувати такий інструмент, коли дані потребують достатньо складних перетворень, як наприклад зєднання по певному полю, перетин або фільтрація за певним критерієм. Процес розробки з використанням мови Pig є в кілька разів швидшим та потребує менше часу та зусиль на відлагодження.

Іншим схожим інструментом є Hive та HCatalog. Це дві взаємоповязані технології, що забезпечують зручний доступ до даних, що збережені в HDFS. HCatalog містить внутрішню базу даних з мета-інформацією про все, що знаходяться на HDFS. Ця інормація використовується при звертанні до даних та їх пошуку. Hive в свою чергу надає SQL-інтерфейс для структурованих даних в системі. Для забезпечення такого доступу, достатньо створити табицю в Hive, в які описати структуру даних та вказати місце розташування. Після цього ми зможемо легко звертатися до цих даних через SQL-інтерфейс. Як і у випадку Pig кожне звертання транслюється в набір Java MapReduce завдань та виконується на кластері. Виконання таких запитів значно поступається традиційним БД в швидкості, але це компенсовується обємами даних, на яких такі запити виконуються. На відміну від звичайних БД така система дозволяє виконувати складні SQL-запити на наборах даних, обємом в кілька десятків терабайт. Для оптимізації зберігання використовуються спеціальні формати з внутрішнім індексуванням та стисненням. На відміну від зберігання в звичайному текстовому форматі, таке зберігання дозволяє зменшити час доступу та обсяг необхідної для зберігання памяті в кілька десятків разів.

Іншим важливим компонентом, який, щоправда не буде використовуватися в цій реалізації системи є NOSQL база даних HBase. Вона забезпечує збереження та доступ майже в реальному часі до обємів даних в кілька десятків терабайт. Це база даних, в яій збереження даних організовано з групуванням по колонках. Характерною рисою її є те, що вона не потребує наперед заданої структури даних, що в ній зберігаються і підтримує її динамічну зміну. Через особливості внутрішньої організації, агрегуючі запити з використанням одної або кількох колонок в такій базі виконуються майже миттєво, незважаючи на обсяги інформації, що в ній збережена.

Компонентом, що все частіше використовується при побудові розподілених систем обробки інформації є фреймворк Spark. Це бібліотека функцій, які дозволяють виконувати обробку даних всіх даних безпосередньо в основній памяті. Як уже було описано в попередньому розділі, основною відмінністю цього інструменту від звичайних MapReduce завдань є широке використання основної памяті як для проміжних так і для основних результатів. Цей фреймворк володіє переліком АРІ для таких мов програмування як Java, Scala та Python, а це надає велику гнучкість в процесі розробки. Моніторинг та координація виконання Spark програм у Hadoop кластері виконується за допомогою менеджера ресурсів YARN. В даній реалізації ця технологія використовуватись не буде. Її використання буде доцільним при побудові системи, що дозволяє обробляти інформацію в реальному часі.

Для взаємодії з СКБД в платформі Hadoop існує інструмент Sqoop. Цей компонент екосистеми надає можливість проводити завантаження інформації з баз даних у розподілену файлову систему. Він володіє командною оболонкою, звертаннями до якої можна специфікувати які колонки з якої таблиці завантажувати в Hadoop. Одновним з основних завдань інструменту Sqoop є скоординувати кількість і частоту запиитів від машин кластера до сервера бази даних. Беручи до уваги, що кластер може складатися з десятків а то й сотень машин, то в ситуації коли всі ці машини почнуть виконувати запити до БД, вона може бути перевантажена і припинити роботу. Для уникнення цього кількість одночасних запитів контролююється даною технологією. Варто сказати, що Sqoop надає достатньо великий перелік можливих операцій для виконання. Так з його допомогою достатньо легко організувати інкременетальне завантаження або вивантаження певних таблиць з БД. В нашій системі цей інструмент буде використовуватися у ролі інтерфейсу для структурованих даних з СКБД. Його виклики відбуватимуться з shell-скриптів, або ж з допомогою функціоналу, який надає програма для координації виконання Oozie.

Для завантаження даних з файлових серверів в нашій системі буде використовуватися стандартний засіб linux для копіювання з віддалених хостів. Це утиліта scp. З її допомогою всі необхідні файли будуть скопійовані на один з компютерів розподіленої системи, а потім засобами файлових команд системи HDFS завантажуватимуться в неї. Такий підхід є простим і надійним. Існують певні програми та утиліти, що надають можливість більш детально конфігурувати цей процес, але для цілей нашої системи такого функціоналу цілком достатньо.

Платформа Hadoop є цілою екосистемою і складається з великої кількості допоміжних частин. Розгортання Hadoop кластера, на наборі машин є зовсім непростим завданням. Потрібно правильно налаштувати всі конфігурації кожної з підсистем. Також часто виникає проблема несумсності деяких версій компонентів між собою. Для спрощення цієї задачі існує інструмент, який дозволяє значно полегшити процес розгортання кластера. Це програма Ambari. Основним її призначенням є керування всіма сервісами, що відносяться до екосистеми Hadoop, та спрощення процесу розгортання кластеру. При її використанні достатньо вказати dns-імена всіх машин кластеру, задати топологію системи, вибрати які сервіси на які машини ми бажаємо встановити, а програма вже сама виконає установку на налаштування цих сервісів. Ця програма володіє зручним веб-інтерфейсом, що надає можливість в інтерактивному режимі керувати кожним сервісом та кожною машиною в межах кластера. Також її зручно використовувати для моніторингу стану системи. В користувацькому інтерфейсі відображається які сервіси запущені на кожній з машин і подається сигнал у випадку зупинки якогось з них. Цей інструмент буде використаний нами на самому початку побудови системи для розгортання та конфігурування кластера.

Одним з найважливіших пунктів при побудові складних систем є організації процесу запуску складних програм та конвеєрів даних. Рідко коли в складних системах обробка інформації обмежується запуском тільки одної підпрограми. Зазвичай таких підпрограм є дуже багато і послідовністю їх запуску відбувається за досить складною логікою. Так, деякі підпрограми повинні виконуватися тільки після завершення попередніх, а інші – з певними часовими інтервалами. Для координації таких послідовностей в екосистемі Hadoop існує інструмент Oozie. Він надає можливість будувати складні послідовності з виконання підпрограм, як наприклад Pig або Hive скриптів, Java програм для виконання MapReduce завдань, Sqoop команд і т.д. Характерною особливістю цього компонента є можливість конфігурування виконання з певними інтервалами, або ж при наявності специфікованих файлів. Це є дуже зручним, коли виникає необхідність запускати послідовності підпрограм тільки у випадку наявності тих чи інших даних. Oozie побудовано у вигляд клієнт-сервер програми. На деяких, або й на всіх машинах кластеру встановлюється Oozie-клієнт, а на одній – сервер. З клієнта на сервер відправляється опис послідовності виконання(у вигляді xml файлу) та параметри для цієї послідовності. Самі програми, які мають виконуватися в межах цієї послідовності повинні розміщуватися на HDFS а шлях до них має бути специфікований у конфігураціному файлі до послідовності виконання. Після отримання завдання, сервер вичитує необхідні параметри, звертається до програм та виконує їх запуск та координацію. Oozie володіє зручним веб-інтерфейсом. Ми можемо слідкувати за станом виконання програм та навіть керувати їх роботою (як наприклад зупиняти, або ж перезапускати певні її елементи). В нашій системі цей компонент використовуватиметься для побудови конвеєра даних. Він координуватиме послідовність та частоту запуску підпрограм в межах нашої програми.

Вся Hadoop платформа була спроектована для кластерів з установленими операційними системами сімейства Linux. Для нашої системи буде використано один з дистрибутивів сімейства Fedora – CentOS. Це безплатний та багатофункціональний дистрибутив. Він поширений в серверних системах та часто використовується в великих організаціях та промисловості. Ця операційна система є одною з найбільш поширених при розгортанні Hadoop кластерів. Для зручності її установки на великій кількості машин та щоб уникнути процесу ручної установку велику кількість разів буде використано інтсрумент автотиматизації та керування кластерами Vagrant. Це система яка значно спрощує процес керування та взаємодії з кластером. Вона дозволяє виконувати розгортання, запуск та зупинку великих кластерів майже миттєво, після їх налаштування. Це великою мірою зменшує час, що необіхний службовому персоналу для керування інфраструктурою системи. Як і всі вище згадані інструменти, цей програмний продукт є абсолютно безплатним, тому може бути використаним для побудови системи сховища даних.

* 1. **Розробка алгоритму формування таблиць вимірів**

Як вже було розглянуто в розділі 1 цієї МКР, в виміро-орієнтованих сховищах даних існує декілька видів таблиць вимірів. Вони відрізняються своєю структурою та порядком формування. В даній системі передбачається наявність компонентів, що виконують побудову шаблонів таблиць вимірів всіх трьох типів. Далі буде наведено алгоритми формування таблиць кожного з видів. Враховуючи що шляхи формування таблиць одного виду є схожі, не зважаючи від предметної галузі, то компонент, що виконує побудову таких таблиць може бути легко конфігурованим для подальшого використання при побудові незалежних сховищ даних.

В розділі 1 було наведено опис трьох типів повільно-змінюваних вимірів. Основна відмінність серед них полягає в переліку даних, що вони містять та правил запису в таку таблицю. Оскільки тип 1 не потребує ніякої алгоритмічної логіки при побудові таблиці, він не буде розглядатися в цій роботі.

Спершу розглянемо алгоритм побудови повільно-змінюваних вимірів типу 2. Нагадаю що в таблицях такого типу, при надходженні нового запису з унікальним ідентифікатором, що вже існує в таблиці, попередній запис має бути заміненим. Також має виконуватися дедублікація записів, тобто в таблиці немає бути записів з однаковими унікальними ідентифікаторами.

Дані для побудови повільно-змінюваних вимірів можуть надходити двома шляхами:

* як підмножина полів в записі таблиці фактів;
* як окремий набір даних, що прямо не звязаний з таблицею фактів.

Незалежно від того яким шляхом прийшли дані для побудови таблиці вимірів, вони володітимуть приблизно ідентичною структурою і саме таку структуру ми візьмемо як початкову точку для побудови таблиці вимірів. Структура інформації, що надійшла, обовязково має містити:

* поле або набір полів, що унікально ідентифікують запис
* поле або набір полів, які зберігають корисну інформацію (для якої власне і формується таблиця вимірів)

Можливим але не обовязковим є атрибут дати створення запису. Цей атрибут є необхідним для побудови таблиць вимірів типу 3 і також певною мірою спрощує побудову таблиць вимірів типу 2.

Алгоритм побудови таблиці вимірів другого типу з вхідних даних наведений в блок-схемі на рис. 3.2.



Рис. 3.2. Блок-схема алгоритму побудови таблиць вимірів другого типу

Як видно з блок-схеми на рис. 3.2. після завантаження всього набору даних першим кроком є групування записів по унікальному ідентифікатору. Результатом виконання цього кроку є набір груп записів з ідентичними ідентифікаторами. В межах однієї групи записи можуть як відрізнятися значеннями інших полів, так і бути ідентичними за цим критерієм. Кожна сформована група записів опрацьовується окремо. Якщо в такій групі присутній тільки один запис, то він одразу зберігається, без додаткових опрацювань. У випадку коли в групі є кілька записів, то спершу виконується порівняння по даті (у випадку якщо неможливо ідентифікувати з якого набору даних є конкретний запис – з тих що уже були збережені раніше, або ж з тих що надійшли). Якщо дати в кількох записах співпадають, то вибирається довільний з записів. У випадку, коли записи в групі відрізняються своїми датами створення, то вибирається той, в кого значення дати є найбільшим, тобто запис, що є новішим за інші. Після такої перевірки вибрані записи зберігаються в таблицю вимірів.

Процес формування повільно-змінюваних вимірів типу 3 є складнішим. Він потребує деяких маніпуляцій з полем часу створення запису, а також введення деякої службової інформації. Як було описано в розділі 1, виміри типу 3 зберігають всі значення кожного з записів, якщо вони змінюються на протязі певного часу. Тобто якщо в записі з унікальним ідентифікатором id1 спершу було значення value1, а через деякий час воно змінилось на value2 то ця зміна має бути збережена в таблиці. Також має бути можливість дізнатись яке значення було розміщено з ідентифікатором id1 в кожний конкретний момент після його створення. Ця інформація може бути корисна при побудові звітів за тривалі періоди часу, а також для збору статистики і відслідковування змін в певній частині предметної області.

Побудова такого виміру потребує одночасного опрацювання як даних що тільки надійшли, так і вже наявних в системі записів з таблиці даного виміру. Це необхідно для коректного встановлення часу використання значення кожного з записів.

Для того щоб отримати можливість звертатися до всіх попередніх значень полів з однаковим ідентифікатором в таблиці вводиться два службових поля – дата створення і дата завершення актуальності. Дата створення відображає час, коли дане значення запису було встановлене. Дата завершення актуальності показує час, коли це значення було змінене на нове. Таким чином, маючи такі додаткові дані ми можемо прослідковувати всю історію змін кожного з записів.

Алгоритм побудови таблиць вимірів типу 3 зображено на блок схемі на рис. 3.3.



Рис. 3.3. Блок-схема алгоритму побудови вимірів типу 3

Як видно з рис. 3.3. при побудові вимірів типу 3, над даними виконується ряд перетворень для того, щоб представити їх в необхідному вигляді. Після формування такого виміру, та таблиці фактів ми можемо тримати необхідний зріз збереженої інформації шляхом зєднання цих двох таблиць. Вибираючи дані з певними конкретними значеннями дат актуальності ми можемо отримати представлення збереженої інформації не тільки в розрізі конкретно обраного виміру, але й з привязкою до часу. Для прикладу, система може формувати аналітичний звіт з врахуванням значень, які були актуальні в певний період в минулому. Це дуже зручна і корисна властивість сховища даних.

В даному розділі не розглядається алгоритм перетворення даних, які належать таблицям фактів. Беручи до уваги те, що сховища даних є предметно орієнтованими, то й трансформації над кожними конкретними екземплярами інформації будуть значною мірою відрізнятися. Саме ця частина бізнес-логіка має бути винесена в окремий модуль, який для кожного окремого сховища даних буде створюватися окрема. Змінюючи та налаштовуючи цей компонент ми виконуватимемо конфігурацію всього сховища даних.

Подібна ситуація є також з частиною системи, яка забезпечує внутрішній та зовнішній інтерфейс системи. Ці інтерфейси повинні формуватися з врахуваннями специфіки та природи даних, назв та кількості полів записів. Як і компонент перетворення таблиць фактів, ці компоненти будуть відрізнятися для кожного конкретного екземпляра сховища даних і тому мають бути сконфігуровані вже безпосередньо в процесі розробки.

Послідовність формування таблиць фактів і таблиць вимірів досить рідко відрізняється в різних предметних областях, тому координатор виконання завдань в системі може повторно використовуватися. Також не залежить від предметної області й процес розгортання Hadoop кластера. Єдина різниця може в кількості машин, з яких складається кластер. Такі системи як Vagrant та Ambari дозволяють абстрагуватися від цієї проблеми та виконувати налаштування та запуск великих кластерів легко і швидко. Все що необхідно для цього – вказати dns-імена всіх машин та вибрати топологію кластера(кількість master- та slave-вузлів).

Як вже було сказано на початку розділу – вся основна програмна логіка буде написана з використанням скриптової мови Pig, яка в свою чергу транслюватиметься в Hadoop-завдання на основі Java-коду.

Система, що була спроектована в даному розіділі може бути розроблена та відлагоджена досить швидко з використанням обмежених розробницьких ресурсів. Проте вона потребує детального розуміння принципу роботи парадигми MapReduce для забезпечення оптимальності та достатньої продуктивності програмного рішення.

## Висновок

В даному розділі проведено проектування системи та вибір технологій для її реалізації. Як було показано вище для створення системи сховища даних може бути використана платформа розподілених обчислень Hadoop, та інші інструменти з цієї екосистеми. Всі запропоновані інструменти легко інтегруються між собою, а це дозволяє значно полегшити процес розробки та відлагодження програмного рішення. Перевагою наведених вище технологій є те, що вони працюють у розподіленому середовищі а тому значно легше горизонтально масштабуються, в порівнянні з класичними інструменами, що розроблені для серверних станцій.

Основним призначенням запропонованого проектного рішення є забезпечення інфраструктури та вихідної архітектури для реалізації конкретного сховища даних. Оскільки сховища даних є предметно-орієтованими і кожна реалізація значно відрізнятиметься від інших, в межах даної роботи не розглядалося алгоритмів побудови конретних сховищ. Було запропонована виключно та програмна логіка, що може бути повторно використана при побудові інших систем. Важливим завданням розроблюваного прототипу є також те, що в ньому вже налаштована інтеграція та взаємодія всіх запропонованих інструментів, тому це значною мірою полегшуватиме розробку майбутніх систем.

Структурна схема та запропоновані алгоритми роботи можуть бути використані для швидкого старту в процесі побудови сховища даних для обраної предметної області. Вони можуть бути легко розширені додатковими алгоритмами роботи, специфічними для обраної предметної області.

# 4. ЕКСПЕРИМЕНТАЛЬНИЙ РОЗДІЛ

В попередніх розділах даної МКР виконувалося дослідження можливості побудови конфігурованого сховища даних на базі розподіленої платформи Hadoop та проектування програмного рішення, на основі виконаних досліджень. Система була спроектована і побудована з урахуванням всіх основних вимог що ставляться до типових сховищ даних, а також з підтримкою простої конфігурації системи в залежності від предметної області, в якій вона буде використовуватися. В даному розділі виконується перевірка коректності роботи системи та її функціональне тестування.

Тестування продуктивності системи не входить в межі даної роботи і потребує наявності достатньо великих обсягів експерементальних даних та відповідної інфраструктури (кластера великої кількості машин). Таке тестування повинне проводитися за наявності деталізованих вимог до сховища даних та з урахуванням інфраструктурних можливостей. Майже лінійна масштабованість платформи Hadoop вже була підтверджена в лабораторних та реальних умовах і є описана в багатьох наукових доповідях.

Даний розділ покриває тільки функціональне тестування системи і виконує валідацію лише базового функціоналу, що є характерним для усіх сховищ даних, незалежно від предметної області. Такий функціонал включає наступні елементи:

* коректне завантаження структурованих даних з СКБД в розподілену файлову систему;
* коректне завантаження окремих файлів з файлових серверів в розподілену файлову систему;
* побудова таблиць фактів з заданої множини вхідних даних;
* побудова таблиць вимірів різних типів з заданої множини вхідних даних;
* надання інтерфейсу до підготованого сховища даних у вигляді, зручному для використання споживачами даних.

Оскільки розроблене рішення є фреймворком і не містить в собі ніяких вбудованих правил обробки, окрім як правил формування таблиць вимірів різних типів, то характер даних для виконання тестування не є важливим. Основною вимогою до цих даних є наявність таких полів:

* дата появи запису;
* унікальний ідентифікатор запису;
* значення яке несе цінність в процесі опрацювання даних (значення яке використовується для побудови аналітичного звіту).

Для валідації системи було згенеровано штучні дані, які відповідають всім вищевказаним вимогам. Кількість тестових даних не є великою, адже вони необхідні тільки для перевірки функціональної правильності системи.

Набір тестових даних складається з чотирьох таблиць наповнених штучно згенерованими даними. В кожній таблиці містяться дані, що використовуватимуться для побудови окремої логічної частини вітрини даних: таблиця фактів, таблиці вимірів першого, другого та третього типів. Ідентичні дані містяться в текстових файлах на локальній файловій системі, яка виконує роль файлового сервера. Для легкості тестування дані були вибрані ідентичними. Записи збережені у форматі розділеному вертикальними лініями (‘|’). Кожне поле в записі відділяється від сусіднього вертикальною лінією. Поширеним форматом є також розділяння комами, але такий спосіб не завжди підходить, адже інколи значення полів можуть містити символи коми. Система сховища даних буде розгорнута у віртуальному середовищі VirtualBox, з використанням уже готового, доступного у вільному доступі, віртуального образа Hortonworks HDP 2.2 Sandbox. База даних з тестовими таблицями працює на хості, засобами якого виконується віртуальне середовище. Для тестових цілей була обрана open-source СКБД MySQL. Текстові файли з тестовими даних знаходяться в локальній файловій системі того ж хоста, на якому працює СКБД.

Запуски виконуються шляхом надсилання послідовностей виконання (описаних в форматі xml, з урахуванням вимог, що ставляться засобом координації виконання Oozie) через командну стрічку на сервер-координатор. Послідовності виконання повинні знаходиться в розподіленій файловій системі(HDFS), а конфігураційний файл з параметрами виконання – на локальній файловій системі, звідки виконується запуск.

Запуски на виконання проводилися в кілька етапів. Спочатку для кожного джерела даних(таблиця в СКБД або текстовий файл на файловійс системі) окремо. Після кожного запуску перевірявся результат кожного з проміжних кроків та послідовності дій загалом. Також виконувався запуск комібонованої послідовності. В межах цієї послідовності виконувалася обробка з усіх джерел даних за один запуск.

Запуски виконувалися тричі для кожного з виду вхідних даних. Таким чином унеможливлювалося випадкове співпадіння отриманих та очікуваних результатів.

Також для перевірки роботи системи використовувалися дані, з завідомо введеними помилками (дані без необхідних полів в записах). Такі запуски виконувалися для підтвердження коректної обробки системою помилкових ситуацій та некоректних файлів.

Результати роботи кожного з запусків порівнювалися з очікуваними результатами. Всі запуски були успішні а результат роботи співпав з тим, який очікувався при запуску.

Після запуску програмного рішення на розподіленій файловій системі було отримано дані, що раніше знаходилися в БД та на окремому файловому сервері. Дані були приведені до вигляду, якого вимагає структура сховища даних та перетворені згідно з описаними вище правилами. Після одержання даних в HDFS, до них був наданий SQL-інтерфейс. Це було виконано за допомогою створення зовнішньої таблиці засобами інструменту Apache Hive. Після цього до даних можна було звертатися як інтерактивно (шляхом викоання SQL-запитів), так і шляхом інтеграції з зовнішніми аналітичними інструментами, які через ODBC-міст зверталися до збережених даних. Швидкість виконання таких запитів для невеликої кількості збереженої інформації є порівняно малою, що пояснюється накладними витратами на кожний з таких запусків. Але перевагою такого рішення є те, що при значному зростанні кількості даних, до яких виконуються запити, час доступу зростає дуже повільно.

Великою перевагою наявності невеликих віртуальних кластерів на локальній машині є можливість тестування окремих частин програми та системи вцілому одразу під час розробки. Функціональне тестування такого коду може відбуватися негайно після його написання, без потреби надсилання його на віддалений кластер. Таке тестування не дозволяє виявити деяких проблем, що можуть зявитися при запуску в справжньому розподіленому середовищі. Тестування в розподіленому середовищі є невідємною частиною процесу розробки таких програмних рішень, як сховища даних і потребує ретельного виконання. Але такий спосіб валідації системи може бути виконаний вже після попередньої перевірки правильності роботи системи локально.

## Висновок

В цьому розділі було проведено тестування та валідацю коректності роботи системи, розробленої під час виконання даної МКР. Основною метою проведених експериментів було встановити правильність виконання поставлених до системи вимог. Було сформульовано основні сценарії для тестування та у відповідності з ними проведено ряд запусків системи з набором штучно-згенерованих даних. Для уникнення можливості випадкового співпадіння отриманих результатів з очікуваними, запуски відбувалися кілька разів декількома способами. Після запуску, результати кожного з проміжних кроків та кінцеві результати порівнювалися з очікуваними. На всіх етапах тестування результати запусків співпадали з очікуваними, що свідчить про коректність роботи системи вцілому.

Дані експерименти не включали перевірки продуктивності, стійкості до навантажень а також інших поширених видів тестувань програмних продуктів. Основною метою було перевірити функціональну правильність виконуваних операцій та підтвердити можливість виконання всіх необхідних операцій збудованою системою. Як можна робити висновок з отриманих результатів – сховища даних на основі платформи Hadoop можуть повністю покрити функціонал, що до цього часу забезпечувався програмними системами на основі СКБД та програм для перетворення даних. Як було описано раніше, такі рішення можуть набагато легше масштабуватися і потребують менших затрат на побудову апаратної інфраструктури.

# 5. ЕКОНОМІЧНИЙ РОЗДІЛі

В межах даної магістерської роботи виконується дослідження шляхів побудови сховищ даних на основі розподіленої системи Hadoop. На даний момент існує декілька шляхів побудови сховищ даних з використанням систем керування базами даних на основі потужних серверних станцій. Недоліком такого підходу є висока вартість необхідних технологій та дуже низька масштабованість таких систем. Розширення таких систем та підвищення їх продуктивності є надзвичайно складним та дорогим. Аналогічні рішення з використанням розподілених обчислювальних систем можуть значно зменшити варітість нарощування ресурсів та полегшити масштабування системи. Платформа Hadoop є порівняно новою і тому раніше не використовувалася в таких цілях. Дослідження можливості побудови та розробка прототипного рішення сховища даних на базі розподіленої платформи є перспективними для подальшого розвитку програмних систем такого типу, та їх потенційної міграції на нові, дешевші технології.

## 5.1 Розрахунок собівартості

До цієї статті належать витрати на основну та додаткову заробітну плату науковому керівнику, студенту, консультанту з питань економіки, обчислені за посадовими окладами та відрядними розцінками для робітників, включаючи преміальні виплати. Вихідні дані наводяться у таблиці 5.1.

Табл. 5.1. Вихідні дані для розрахунку витрат на оплату праці

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Посада виконавців | Місячний оклад, грн. | Середньоденна ставка, грн/дн |
| 1 | Керівник МКР, доцент | 3000 | 142,86 |
| 2 | Консультант з економіки | 2500 | 119,05 |
| 4 | Студент | 600 | 28,57 |

Витрати на оплату праці розробників проекту визначаються за формулою:

 (5.1)

де *nij* – чисельність розробників *і-*ої спеціальності *j*-го тарифного розряду, які приймають участь в проектуванні, чол.; *tij* – час, котрий затрачений на розробку проекту співробітника *і*-ої спеціальності *j*-го тарифного розряду, днів; *Cij* – денна заробітна плата *і*-ої спеціальності *j*-го тарифного розряду, грн., яка визначається за формулою:

 (5.2)

де *Cij*– основна місячна заробітна плата розробника *і*-ої спеціальності *j*-го тарифного розряду, грн.; *h* – коефіцієнт, що визначає розмір додаткової заробітної плати (при умові наявності доплат); *р* – середня кількість робочих днів у місяці (приймаємо 21 р.д.).

Табл. 5.2. Розрахунок витрат на оплату праці

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № п/п | Посада виконавців | Час розробки, дні | Денна заробітна плата, грн | Витрати на розробку, грн |
| 1 | Керівник МКР, доцент | 10 | 142,86 | 1428,6 |
| 2 | Консультант з економіки | 1 | 119,05 | 119,05 |
| 3 | Студент | 60 | 28,57 | 1714,29 |
| Разом | | | | 3261,94 |

Величину відрахувань у спеціальні державні фонди визначають у процентному співвідношенні від суми основної та додаткової заробітної плати. Згідно діючого нормативного законодавства сума відрахувань у спеціальні державні фонди складає 36,2%від суми заробітної плати:

Вф=0,362\*З (5.4)

Вф=0,362\*3261,94=1180,82 грн.

Табл. 5.3. Розрахунок витрат на куповані вироби

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № п/п | Найменування купованих виробів | Одиниця виміру | Ціна на одиницю виміру, грн | Кількість купованих виробів | Сума, грн | Транспортні витрати (10% від суми) | Загальна сума,  Грн. |
| 1 | Папір (формат А4) | уп | 45,0 | 1 | 45,00 | 9,0 | 49,5 |
| 2 | Ручка кулькова | шт | 2,0 | 2 | 4,00 | 0,4 | 4,40 |
| 3 | Олівець простий | шт | 1,50 | 2 | 3,00 | 0,3 | 3,30 |
| 4 | Диски CD-R | шт | 2,0 | 2 | 4,00 | 0,4 | 4,40 |
| 5 | Зошит, 96 арк | шт | 3,50 | 1 | 3,50 | 0,35 | 3,85 |
| 6 | Тонер для принтера | уп | 20 | 1 | 20 | 2,0 | 22,0 |
| Разом | | | | | | | 87,45 |

Витрати на використання комп’ютерної техніки включають витрати на амортизацію комп’ютерної техніки, витрати на користування програмним забезпеченням, витрати на електроенергію, що споживається комп’ютером. За даними обчислювального центру НУ «Львівська політехніка» для ЕОМ типу ІВМ РС/АТХ вартість години роботи становить 4,5 грн. Середній щоденний час роботи на комп’ютері – 4 години. Розрахунок витрат на використання комп’ютерної техніки приведений в таблиці 5.4.

Табл. 5.4. Розрахунок витрат на використання комп’ютерної техніки

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| №  п/п | Назва етапів робіт, при виконанні  яких використовується  комп’ютер | Час використання  комп’ютера | | Витрати на використання комп’ютера,  Грн |
| днів | годин |
| 1 | Проведення досліджень та оформлення їх результатів | 30 | 120 | 540 |
| 2 | Оформлення розділу “Економіка” | 2 | 8 | 36 |
| 3 | Оформлення МКР | 16 | 64 | 288 |
| Разом | | 48 | 192 | 864 |

В процесі досліджень не використовувалося спеціальне обладнання.

Накладні витрати проектних організацій включають три групи видатків: витрати на управління, загальногосподарські витрати, невиробничі витрати. Вони розраховуються за встановленими відсотками до витрат на оплату праці. Середньостатистичний відсоток накладних витрат в організації складає 150%.

Н = 1,5\*З (5.5)

Н = 1,5\*3261,94 = 4892,91 (грн.)

Інші витратиє витратами, які не враховані в інших статтях. Вони становлять 10% від заробітної плати:

Ів =0,1\*З (5.6)

Ів=0,1\*3261,94=326,19 грн.

На основі отриманих даних виконуємо калькуляцію планової собівартості загалом по НДР і зводимо дані в таблицю 5.5:

Табл. 5.5. Кошторис витрат на виконання НДР

|  |  |  |
| --- | --- | --- |
| № п/п | Найменування елементів витрат | Сума витрат, грн |
| 1 | Витрати на оплату праці | 3216,85 |
| 2 | Відрахування у спеціальні державні фонди | 1180,82 |
| 3 | Витрати на куповані вироби | 87,45 |
| 4 | Витрати на використання комп’ютерної техніки | 864 |
| 5 | Накладні витрати | 4892,91 |
| 6 | Інші витрати | 326,19 |
| Разом | | 10568.19 |

К=3 + Вф + Кв + Об + Н + Ів (5.7)

К =3216,85+1180,82 +87,45+864+4892,91+326,19 =10568,19 грн.

**5.2 Розрахунок договірної ціни та прибутку НДР**

Величина договірної ціни повинна встановлюватися з врахуванням ефективності, якості і термінів виконання НДР на рівні, який відповідає економічним інтересам замовника (споживача) і виконавця.

Договірна ціна НДР встановлюється по домовленості між замовником та виконавцем і попередньо розраховується за формулою:

Ц=С\*(1+р), (5.8)

де Ц – договірна ціна, С – собівартість проведення НДР, р – рівень рентабельності витрат у виконавця НДР.

Тоді очікуваний прибуток від реалізації НДР розраховують за формулою:

П = Цд- С, (5.9)

де П – очікуваний прибуток від виконання НДР.

Для даної НДР ціна визначена як:

Цд = С + 0,3\*С = 1,3\*С = 1,3\*10568,19 = 13738,64 грн.

Тоді прибуток буде рівним:

П = 1,3\*C – C = 0,3\*C = 0,3 \* 10568,19 = 3170,45 (грн)

* 1. **Оцінка наукової та науково-технічної результативності НДР**

Результатом НДР є досягнення наукового, науково-технічного, економічного або соціального ефекту. У даному підрозділі оцінено рівень наукового та науково-технічного ефектів.

Науковий ефект характеризує отримання нових знань та відображає приріст інформації, призначеної для внутрінаукового застосування. Науково-технічний ефект характеризує можливість використання результатів виконуваних досліджень у інших НДР чи ДКР і забезпечує отримання інформації, необхідної для створення нової техніки.

Оцінка наукової та науково-технічної ефективності для НДР проводиться за допомогою коефіцієнтів, які обчислюються за формулами:

(5.10)

(5.11)

де та – відповідно, коефіцієнти наукової та науково-технічної результативності; – коефіцієнт значимості *і*-го фактору, використовуваного для оцінки; – коефіцієнт досягнутого рівня *і*-го фактору; n, m – відповідно, кількість факторів наукової та науково-технічної результативності.

При оцінці наукової і науково-технічної результативності використовуються різні фактори, які впливають на її кількісну оцінку. В якості факторів при оцінці наукової ефективності можуть бути прийняті наступні: новизна отриманих чи прогнозованих результатів; глибина наукового опрацювання; ступінь вірогідності успіху (при незавершеності роботи). В якості факторів при оцінці науково-технічної результативності можуть застосовуватися: перспективність використання результатів; масштаб можливої реалізації результатів; завершеність отриманих результатів.

По кожному із факторів експертним шляхом встановлюється числове значення коефіцієнта значимості. При цьому, сума цих коефіцієнтів повинна бути рівною 1. Коефіцієнт досягнутого рівня фактору також встановлюється експертним шляхом, а його числове значення визначається з урахуванням якості ознаки фактору і його характеристики. При цьому його значення не перевищує 1.

Максимально можливе значення коефіцієнтів наукової та науково-технічної ефективності рівне 1. Чим ближчі їх значення до 1-і, тим більша результативність НДР, яка проводиться.

Табл. 5.6. Характеристики факторів та ознак наукової

результативності НДР

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Фактор наукової результативності | Коефіцієнт значимості фактору | Якість фактора | Характеристика фактора | Коефіцієнт досягнутого рівня |
| Новизна отриманих чи прогнозованих результатів | 0,5 | Середня | Встановлені декотрі загальні закономірності, методи, способи, які дають змогу створити принципово нові види техніки | 0,7 |
| Глибина наукового опрацювання | 0,35 | Середня | Складність теоретичних розрахунків невелика, результати перевірені на обмеженій кількості експериментальних даних | 0,6 |
| Ступінь вірогідності успіху | 0,15 | Висока | Успіх вельми можливий, є велика імовірність позитивного вирішення поставлених задач | 1,0 |

Відповідні дані для виконуваної НДР приведені в таблиці 5.7.

Таблиця 5.7. Характеристики факторів та ознак науково-технічної результативності НДР

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Фактор науково-технічної результативності | Коефіцієнт значимості фактору | Якість фактора | Характеристика фактора | Коефіцієнт досягнутого рівня |
| Перспективність використання результатів | 0,5 | Важлива | Результати будуть використані в конкретному науковому напрямі при розробці нових технічних рішень, націлених на суттєве підвищення продуктивності праці | 0,8 |
| Масштаб можливої реалізації результатів | 0,3 | Галузевий | Час реалізації: до трьох років | 0,8 |
| Завершеність отриманих результатів | 0,2 | Середня | Технічне завдання на прикладні НДР чи ДКР | 0,8 |

Згідно із наведенеми даними розраховуємо коефіцієнти:

* наукової ефективності:

;

* науково-технічної ефективності:

**Висновок**

Під час виконання даного розділу було розраховано необхідні витрати для проведення НДР, прибуток та договірну ціну. Також оцінено наукову та науково-технічну результативність НДР.

Дані дослідження були призначення для визначення та оцінки можливості побудови сховищ даних з використанням новітніх технологій, зокрема розподіленої платформи обчислень Hadoop. Використання цих технологій дозволить значною мірою покращити масштабованість подібних систем і таким чином полегшити процес їх удосконалення. Результатом цього буде значне зменшення вартості інфраструктури для системи. Дослідження показало, що використання таких технологій при побудові системи є не тільки можливим але й дозволяє покращити деякі з її характеристик.

# ВИСНОВКИ

Під час виконання даної роботи було проведено дослідження методів побудови конфігурованих сховищ даних на основі платформи Hadoop. Як результат дослідження було запропоновано кілька шляхів побудови таких сховищ даних. Як критерії для порівняння запропонованих методів вибиралися особливості, що характерні системам сховищ даних. Дослідження кожного з методів включало визначення всіх основних переваг та недоліків. Також результатом даної роботи є архітектурна діаграма програмного рішення та прототип такої системи. В розробленому прототипі весь функціонал, який може бути використаний для розробки схожих систем був відділений від предметно-залежної частини. Це дозволяє значно полегшити процес розробки конкретного сховища даних шляхом конфігурації та доповнення розробленого прототипу.

В роботі повністю виконано поставлене завдання та детально описано процес його виконання. Дана робота може бути використана для прискорення процесу побудови та реалізації сховищ даних. Нові шляхи реалізації таких програмних продуктів, що запропоновані в цій роботі, можуть значно полегшити проектування предметно-орієнтованих систем збереження даних.

# СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Ralph Kimball, Margy Ross. The Data Warehouse Toolkit, 2nd Edition, 2002. - 446 с.
2. Tom White. Hadoop: The Definitive Guide, 3rd Edition, O'Reilly Media / Yahoo Press, 2012. - 668 c.
3. Konstantin Shvachko, Hairong Kuag, The Hadoop Distributed File System, Стаття, 2004, Yahoo! Sunnyvale, California, USA.
4. В.М.Олійник - Інформаційні системи і технології у фінансах. Конспект лекцій, Видавництво СумДУ, 2010
5. Jeffrey Dean and Sanjay Ghemawat, MapReduce: Simplified Data Processing on Large Clusters, Стаття, 2004, Google, Inc.
6. Matei Zaharia, Mosharaf Chowdhury Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing, стаття, 2012, University of California, Berkeley
7. James Kinley, The Lambda architecture: principles for architecting realtime Big Data systems [Електронний ресурс] Режим доступу:

http://jameskinley.tumblr.com/post/37398560534/the-lambda-architecture-principles-for

1. Hortonworks Data Platform 2.1 Official Documentation [Електронний ресурс] Режим доступу:

http://docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.1.2/index.html

1. Lars George - HBase: The Definitive Guide, O'Reilly Media, 2011, 556с
2. Cloudera Product Documentation [Електронний ресурс] Режим доступу: http://www.cloudera.com/content/cloudera/en/documentation.html
3. Alan Gates - Programming Pig. Dataflow Scripting with Hadoop, O'Reilly Media, 2011, 224с
4. Edward Capriolo, Dean Wampler, Jason Rutherglen - Programming Hive. Data Warehouse and Query Language for Hadoop, O'Reilly Media, 2012, 352с
5. Holden Karau, Andy Konwinski, Patrick Wendell, Matei Zaharia - Learning Spark. Lightning-Fast Big Data Analytics, O'Reilly Media, 2014, 300с
6. Kathleen Ting, Jarek Jarcec Cecho - Apache Sqoop Cookbook. Unlocking Hadoop for Your Relational Database, O'Reilly Media, 2013, 94с

# ДОДАТКИ

Опис запуску завантаження з бази даних

<workflow-app name="ExecSqoopJob" xmlns="uri:oozie:workflow:0.4">

<start to="execSqoopJob"/>

<action name="execSqoopJob">

<sqoop xmlns="uri:oozie:sqoop-action:0.2">

<job-tracker>${jobTracker}</job-tracker>

<name-node>${nameNode}</name-node>

<arg>job</arg>

<arg>--exec</arg>

<arg>${jobName}</arg>

<arg>--</arg>

<arg>--target-dir</arg>

<arg>${targetDir}/${date}</arg>

<arg>--fields-terminated-by</arg>

<arg>|</arg>

</sqoop>

<ok to="exposeHiveTable"/>

<error to="kill"/>

</action>

<action name="exposeHiveTable">

<hive xmlns="uri:oozie:hive-action:0.2">

<job-tracker>${jobTracker}</job-tracker>

<name-node>${nameNode}</name-node>

<job-xml>hive/hive-site.xml</job-xml>

<configuration>

<property>

<name>mapred.job.queue.name</name>

<value>${queueName}</value>

</property>

</configuration>

<script>hive/${hiveScript}</script>

<param>dbName=${dbName}</param>

<param>dbLocation=${dbLocation}</param>

<param>tableName=${tableName}</param>

<param>tableLocation=${tableLocation}</param>

<param>rawDataLocation=${rawDataLocation}</param>

<param>date=${date}</param>

</hive>

<ok to="end"/>

<error to="kill"/>

</action>

<kill name="kill">

<message>Action failed, error message[${wf:errorMessage(wf:lastErrorNode())}]</message>

</kill>

<end name="end"/>

</workflow-app>

Опис запуску завантаження з бази даних

<workflow-app name="LoadRDBMSSqoopJob" xmlns="uri:oozie:workflow:0.4">

<start to="facilitiesSqoopJob"/>

<action name="facilitiesSqoopJob">

<sqoop xmlns="uri:oozie:sqoop-action:0.2">

<job-tracker>${jobTracker}</job-tracker>

<name-node>${nameNode}</name-node>

<arg>import</arg>

<arg>--connect</arg>

<arg>${database}</arg>

<arg>--username</arg>

<arg>${username}</arg>

<arg>--password-file</arg>

<arg>${passwordFile}</arg>

<arg>--table</arg>

<arg>${table}</arg>

<arg>--target-dir</arg>

<arg>${targetDir}/${date}</arg>

<arg>-m</arg>

<arg>1</arg>

<arg>--fields-terminated-by</arg>

<arg>|</arg>

<arg>--null-string</arg>

<arg>'\\N'</arg>

<arg>--null-non-string</arg>

<arg>'\\N'</arg>

<archive>${flowDir}/lib/mysql-connector-java-5.1.30.jar#mysql-connector-java-5.1.30.jar</archive>

</sqoop>

<ok to="exposeHiveTable"/>

<error to="kill"/>

</action>

<action name="exposeHiveTable">

<hive xmlns="uri:oozie:hive-action:0.2">

<job-tracker>${jobTracker}</job-tracker>

<name-node>${nameNode}</name-node>

<job-xml>hive/hive-site.xml</job-xml>

<configuration>

<property>

<name>mapred.job.queue.name</name>

<value>${queueName}</value>

</property>

</configuration>

<script>hive/${hiveScript}</script>

<param>dbName=${dbName}</param>

<param>dbLocation=${dbLocation}</param>

<param>tableName=${tableName}</param>

<param>tableLocation=${tableLocation}</param>

<param>rawDataLocation=${rawDataLocation}</param>

<param>date=${date}</param>

</hive>

<ok to="end"/>

<error to="kill"/>

</action>

<kill name="kill">

<message>Action failed, error message[${wf:errorMessage(wf:lastErrorNode())}]</message>

</kill>

<end name="end"/>

</workflow-app>

Файл з параметрами для запуску послідовностей

nameNode=hdfs://localhost.localdomain:8020

jobTracker=localhost.localdomain:8032

queueName=default

oozie.use.system.libpath=true

oozie.libpath=/user/oozie/share/lib

homeDir=${nameNode}/user/${user.name}

appDir=${homeDir}/ingestion

appName=raw-1.0-app

flowDir=${appDir}/flows/${appName}

dataDir=${appDir}/raw

oozie.wf.application.path=${flowDir}/ LoadRDBMSSqoopJob.xml

#oozie.coord.application.path=${flowDir}/coord.xml

# workflow

database=jdbc:mysql://localhost/data

username=sqoop

passwordFile=${flowDir}/sqoop.password

table=table1

targetDir=${dataDir}/${table}

date=20131024

Скрипт для початково перетворення отриманих даних і збереження з партиціюванням

REGISTER ${jarName};

DEFINE GetLastDayOfMonth com.ingestion.rdbms.pig.GetLastDayOfMonth();

raw = LOAD '${dbName}.${srcTable}' USING org.apache.hive.hcatalog.pig.HCatLoader();

rawWithPartitioningColumn = FOREACH raw GENERATE \*, GetLastDayOfMonth(create\_date\_time) AS effective\_date;

rawWithoutCurrentMonthsRecords = FILTER rawWithPartitioningColumn BY (int)effective\_date<(int)'${date}';

STORE rawWithoutCurrentMonthsRecords INTO '${dbName}.${destTable}' USING org.apache.hive.hcatalog.pig.HCatStorer();

Додаткова функція що використовується в скрипті початкового перетворення

package com.ingestion.rdbms.pig;

import java.io.IOException;

import java.util.Calendar;

import org.apache.pig.EvalFunc;

import org.apache.pig.data.Tuple;

import org.joda.time.DateTime;

import org.joda.time.format.DateTimeFormat;

import org.joda.time.format.DateTimeFormatter;

public class GetLastDayOfMonth extends EvalFunc<String> {

private static final DateTimeFormatter OUTPUT\_DATE\_FORMATTER = DateTimeFormat.forPattern("yyyyMMdd");

@Override

public String exec(Tuple input) throws IOException {

Calendar dateOfRow = Calendar.getInstance();

dateOfRow.setTime(((DateTime)input.get(0)).toDate());

dateOfRow.add(Calendar.MONTH, 1);

dateOfRow.set(Calendar.DAY\_OF\_MONTH, 1);

dateOfRow.add(Calendar.DATE, -1);

return OUTPUT\_DATE\_FORMATTER.print(dateOfRow.getTimeInMillis());

}

}

Створення SQL-інтерфейсу для отриманих даних

CREATE DATABASE IF NOT EXISTS ${dbName} LOCATION '${dbLocation}';

CREATE EXTERNAL TABLE IF NOT EXISTS ${dbName}.${tableName} (

id INT,

state STRING

)

PARTITIONED BY(effective\_date STRING)

ROW FORMAT DELIMITED FIELDS TERMINATED BY '|'

STORED AS TEXTFILE

LOCATION '${tableLocation}';

USE ${dbName};

ALTER TABLE ${tableName} ADD PARTITION (effective\_date='${date}') LOCATION '${rawDataLocation}/${date}';

Опис послідовності побудови виміру типу 2 з отриманих з БД даних

<workflow-app name="Raw2Stg" xmlns="uri:oozie:workflow:0.4">

<start to="createStgSchema"/>

<action name="createStgSchema">

<hive xmlns="uri:oozie:hive-action:0.2">

<job-tracker>${jobTracker}</job-tracker>

<name-node>${nameNode}</name-node>

<job-xml>hive/hive-site.xml</job-xml>

<configuration>

<property>

<name>mapred.job.queue.name</name>

<value>${queueName}</value>

</property>

</configuration>

<script>hive/${createStgSchemaScript}</script>

<param>dbName=${dbName}</param>

<param>stgTable=${stgTable}</param>

<param>stgLocation=${stgLocation}</param>

</hive>

<ok to="createTmpSchema"/>

<error to="kill"/>

</action>

<action name="createTmpSchema">

<hive xmlns="uri:oozie:hive-action:0.2">

<job-tracker>${jobTracker}</job-tracker>

<name-node>${nameNode}</name-node>

<job-xml>hive/hive-site.xml</job-xml>

<configuration>

<property>

<name>mapred.job.queue.name</name>

<value>${queueName}</value>

</property>

</configuration>

<script>hive/${createTmpSchemaScript}</script>

<param>dbName=${dbName}</param>

<param>tmpTable=${tmpTable}</param>

<param>tmpTableLocation=${tmpTableLocation}</param>

</hive>

<ok to="calculateTimeFrameBound"/>

<error to="kill"/>

</action>

<action name="calculateTimeFrameBound">

<java>

<job-tracker>${jobTracker}</job-tracker>

<name-node>${nameNode}</name-node>

<main-class>com.ingestion.utils.TimeFrameLowerBoundPreparator</main-class>

<arg>${currentDate}</arg>

<arg>${timeFrameWidth}</arg>

<capture-output/>

</java>

<ok to="merge"/>

<error to="kill"/>

</action>

<action name="merge">

<pig>

<job-tracker>${jobTracker}</job-tracker>

<name-node>${nameNode}</name-node>

<job-xml>hive/hive-site.xml</job-xml>

<configuration>

<property>

<name>mapred.job.queue.name</name>

<value>${queueName}</value>

</property>

</configuration>

<script>pig/${mergeScript}</script>

<param>dbName=${dbName}</param>

<param>rawTableName=${rawTable}</param>

<param>rawPartitionValue=${rawPartitionValue}</param>

<param>stgTableName=${stgTable}</param>

<param>tmpTableName=${tmpTable}</param>

<param>timeFrameLowerBound=${wf:actionData('calculateTimeFrameBound')['date']}</param>

<param>currentDateTime=${currentDateTime}</param>

<param>format=${format}</param>

</pig>

<ok to="updateStg"/>

<error to="kill"/>

</action>

<action name="updateStg">

<hive xmlns="uri:oozie:hive-action:0.2">

<job-tracker>${jobTracker}</job-tracker>

<name-node>${nameNode}</name-node>

<job-xml>hive/hive-site.xml</job-xml>

<configuration>

<property>

<name>mapred.job.queue.name</name>

<value>${queueName}</value>

</property>

</configuration>

<script>hive/${updateStgScript}</script>

<param>dbName=${dbName}</param>

<param>tmpTable=${tmpTable}</param>

</hive>

<ok to="end"/>

<error to="kill"/>

</action>

<kill name="kill">

<message>Action failed, error message[${wf:errorMessage(wf:lastErrorNode())}]</message>

</kill>

<end name="end"/>

</workflow-app>

Параметри для запуску послідовності виконання

nameNode=hdfs://sandbox.hortonworks.com:8020

jobTracker=sandbox:8050

queueName=default

oozie.use.system.libpath=true

oozie.libpath=/user/oozie/share/lib

homeDir=${nameNode}/user/${user.name}

appDir=${homeDir}/ingestion

appName=stg-1.0-app

flowsDir=${appDir}/flows/${appName}

stgDir=${appDir}/stg

oozie.wf.application.path=${flowsDir}/raw2stg.xml

# create stg schema

createStgSchemaScript=stg\_ddl.hql

stgTable=stg

stgLocation=${stgDir}/stg

# create tmp schema

createTmpSchemaScript=tmp\_ddl.hql

tmpTable=tmp

tmpTableLocation=${stgDir}/tmp

# calculate time frame bound

currentDate=20131029

timeFrameWidth=65

# merge

mergeScript=mergeRaw2Stg.pig

rawTable=raw

rawPartitionValue=20131024

currentDateTime=2014-07-07 00:00:00

format=yyyy-MM-dd HH:mm:ss

# update stg

updateStgScript=update\_stg.hql

Скрипт початкового завантаження

raw = LOAD '${dbName}.${rawTableName}' using org.apache.hive.hcatalog.pig.HCatLoader();

A = FOREACH raw GENERATE

GetYear(create\_date\_time) AS year,

GetMonth(create\_date\_time) AS month,

account\_id;

STORE A INTO '${dbName}.${tmpTableName}' USING org.apache.hive.hcatalog.pig.HCatStorer();

Скрипт обєднання існуючих і нових даних у вимірі

-- LOAD PARTITIONS

raw = LOAD '${dbName}.${rawTableName}' using org.apache.hive.hcatalog.pig.HCatLoader();

part = FILTER raw BY effective\_date > '${timeFrameLowerBound}' AND MonthsBetween(create\_date\_time, ToDate('${currentDateTime}','${format}')) >= 0 AND MonthsBetween(create\_date\_time, ToDate('${currentDateTime}','${format}')) <= 1;;

-- LOAD STAGING

stg = LOAD '${dbName}.${stgTableName}' using org.apache.hive.hcatalog.pig.HCatLoader();

-- NORMALIZE SCHEMAS

A = FOREACH part GENERATE account\_id,

account\_balance,

create\_date\_time,

update\_date\_time;

B = FOREACH stg GENERATE account\_id,

create\_date\_time,

update\_date\_time;

-- MERGE

grouped = COGROUP A BY (account\_id, ToString(create\_date\_time, '${format}')), B BY (account\_id, ToString(create\_date\_time, '${format}'));

merged = FOREACH grouped {

d1 = FOREACH A GENERATE ToMilliSeconds(update\_date\_time) AS ms:long;

d2 = FOREACH B GENERATE ToMilliSeconds(update\_date\_time) AS ms:long;

result = CASE

WHEN IsEmpty(A) THEN B

WHEN IsEmpty(B) THEN A

WHEN MAX(d1.ms) >= MAX(d2.ms) THEN A ELSE B

END;

GENERATE FLATTEN(result);

}

-- ADD PARTITION FIELDS

tmp = FOREACH merged GENERATE

GetYear(B::create\_date\_time) AS year,

GetMonth(B::create\_date\_time) AS month,

B::account\_id AS account\_id;

STORE tmp INTO '${dbName}.${tmpTableName}' USING org.apache.hive.hcatalog.pig.HCatStorer();

Допоміжний клас для визначення обмежуючих часових меж

package com.ingestion.rdbms.utils;

import java.io.File;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.OutputStream;

import java.util.Calendar;

import java.util.Properties;

import org.joda.time.DateTime;

import org.joda.time.format.DateTimeFormat;

import org.joda.time.format.DateTimeFormatter;

public class TimeFrameLowerBoundPreparator {

public static final DateTimeFormatter DATE\_KEY\_FORMATTER = DateTimeFormat.forPattern("yyyyMMdd");

private static DateTime parseDateByNaturalKey(String date) {

return DATE\_KEY\_FORMATTER.parseDateTime(date);

}

public static void main(String[] args) throws IOException{

int timeFrameWidth = Integer.parseInt(args[1]);

Calendar lowerBound = Calendar.getInstance();

lowerBound.setTimeInMillis(parseDateByNaturalKey(args[0]).getMillis());

lowerBound.add(Calendar.DAY\_OF\_MONTH, -timeFrameWidth);

String ooziePropFileName = System.getProperty("oozie.action.output.properties");

File file = new File(ooziePropFileName);

Properties props = new Properties();

props.setProperty("date", DATE\_KEY\_FORMATTER.print(lowerBound.getTimeInMillis()));

OutputStream os = new FileOutputStream(file);

props.store(os, "");

os.close();

}

}

Створення SQL-інтерфейсу для проміжних та результуючих даних

USE ${dbName};

CREATE EXTERNAL TABLE IF NOT EXISTS ${rawTable} (

account\_id STRING,

create\_date\_time TIMESTAMP,

update\_date\_time TIMESTAMP)

COMMENT 'This is the raw payment adjustment table'

PARTITIONED BY(effective\_date STRING)

ROW FORMAT DELIMITED FIELDS TERMINATED BY '|'

STORED AS TEXTFILE

LOCATION '${rawLocation}';

USE ${dbName};

LOAD DATA INPATH '${PathOne}' INTO TABLE ${rawTable} PARTITION(effective\_date='${DateOne}');

LOAD DATA INPATH '${PathTwo}' INTO TABLE ${rawTable} PARTITION(effective\_date='${DateTwo}');

set hive.exec.dynamic.partition.mode=nonstrict;

USE ${dbName};

FROM ${tmpTable} tmp

INSERT OVERWRITE TABLE stg PARTITION(year, month)

SELECT account\_id, create\_date\_time, update\_date\_time, year, month DISTRIBUTE BY year, month;

Опис послідовності виконання перетворення отриманих даних в вимір типу 3

<workflow-app name="Type3" xmlns="uri:oozie:workflow:0.4">

<start to="createStgSchema"/>

<action name="createStgSchema">

<hive xmlns="uri:oozie:hive-action:0.2">

<job-tracker>${jobTracker}</job-tracker>

<name-node>${nameNode}</name-node>

<job-xml>hive/hive-site.xml</job-xml>

<configuration>

<property>

<name>mapred.job.queue.name</name>

<value>${queueName}</value>

</property>

</configuration>

<script>hive/${createStgSchemaScript}</script>

<param>dbName=${dbName}</param>

<param>stgTable=${stgTable}</param>

<param>stgLocation=${stgLocation}</param>

</hive>

<ok to="merge"/>

<error to="kill"/>

</action>

<action name="merge">

<pig>

<job-tracker>${jobTracker}</job-tracker>

<name-node>${nameNode}</name-node>

<job-xml>hive/hive-site.xml</job-xml>

<configuration>

<property>

<name>mapred.job.queue.name</name>

<value>${queueName}</value>

</property>

</configuration>

<script>pig/${mergeScript}</script>

<param>dbName=${dbName}</param>

<param>rawTableName=${rawTable}</param>

<param>output=${stgLocation}\_${date}</param>

<file>${flowsDir}/lib/datafu-1.2.0.jar#datafu-1.2.0.jar</file>

</pig>

<ok to="updateStg"/>

<error to="kill"/>

</action>

<action name="updateStg">

<hive xmlns="uri:oozie:hive-action:0.2">

<job-tracker>${jobTracker}</job-tracker>

<name-node>${nameNode}</name-node>

<job-xml>hive/hive-site.xml</job-xml>

<configuration>

<property>

<name>mapred.job.queue.name</name>

<value>${queueName}</value>

</property>

</configuration>

<script>hive/${updateStgScript}</script>

<param>dbName=${dbName}</param>

<param>stgTable=${stgTable}</param>

<param>newStgLocation=${stgLocation}\_${date}</param>

</hive>

<ok to="end"/>

<error to="kill"/>

</action>

<kill name="kill">

<message>Action failed, error message[${wf:errorMessage(wf:lastErrorNode())}]</message>

</kill>

<end name="end"/>

</workflow-app>

Файл з параметрами для запуску послідовності виконання

nameNode=hdfs://sandbox.hortonworks.com:8020

jobTracker=sandbox:8050

queueName=default

oozie.use.system.libpath=true

oozie.libpath=/user/oozie/share/lib

homeDir=${nameNode}/user/${user.name}

appDir=${homeDir}/ingestion

appName=stg-1.0-app

flowsDir=${appDir}/flows/${appName}

rawDir=${appDir}/raw

oozie.wf.application.path=${flowsDir}/type3.xml

# create database

createDatabaseScript=database\_ddl.hql

dbLocation=${appDir}/warehouse

# create raw schema

createRawSchemaScript=raw\_ddl.hql

rawTable=raw

rawLocation=${rawDir}/raw

# load raw data

loadRawDataScript=raw\_load.hql

PathOne=/user/${user.name}/input/20131024.txt

DateOne=20131024

PathTwo=/user/${user.name}/input/20131025.txt

DateTwo=20131025

PathThree=/user/${user.name}/input/20131028.txt

DateThree=20131028

Скрипт побудови вимірів типу 3

REGISTER datafu-1.2.0.jar

DEFINE Enumerate0 datafu.pig.bags.Enumerate();

DEFINE Enumerate1 datafu.pig.bags.Enumerate('1');

DEFINE BagLeftOuterJoin datafu.pig.bags.BagLeftOuterJoin();

DEFINE COALESCE datafu.pig.util.Coalesce();

raw = LOAD '${dbName}.${rawTableName}' using org.apache.hive.hcatalog.pig.HCatLoader();

grouped = GROUP raw BY (facility\_code, trans\_code);

joined = FOREACH grouped {

sorted = ORDER raw BY create\_date\_time ASC, update\_date\_time ASC;

e0 = Enumerate0(sorted);

e1 = Enumerate1(sorted);

GENERATE group as id, BagLeftOuterJoin(e1,'i',e0,'i') as items;

}

projected = FOREACH joined {

A = FOREACH items GENERATE

$0 AS facility\_code,

$1 AS trans\_code,

$2 AS trans\_desc,

$3 AS trans\_category,

$4 AS payor,

$5 AS is\_patient\_portion,

$6 AS active\_flag,

$8 AS from\_date\_time,

-- CASE

-- WHEN $19 IS NOT NULL THEN SubtractDuration(ToDate($19, 'yyyy-MM-dd HH:mm:ss'),'P1D') ELSE ToDate('9999-12-31 00:00:00','yyyy-MM-dd HH:mm:ss')

-- END;

COALESCE ($19, ToDate('9999-12-31 00:00:00','yyyy-MM-dd HH:mm:ss')) as to\_date\_time;

GENERATE FLATTEN(A);

}

filtered = FILTER projected BY A::from\_date\_time != A::to\_date\_time;

result = FOREACH filtered GENERATE

$0 AS facility\_code,

$1 AS trans\_code,

$2 AS trans\_desc,

$3 AS trans\_category,

$4 AS payor,

$5 AS is\_patient\_portion,

$6 AS active\_flag,

$7 AS from\_date\_time,

$8 AS to\_date\_time;

STORE result INTO '${output}' USING PigStorage('|');

Створення SQL-інтерфейсу для виміру типу 3

USE ${dbName};

CREATE EXTERNAL TABLE IF NOT EXISTS ${stgTable} (

facility\_code STRING,

trans\_code STRING,

trans\_desc STRING,

trans\_category STRING,

payor STRING,

is\_patient\_portion STRING,

active\_flag STRING,

from\_date\_time TIMESTAMP,

to\_date\_time TIMESTAMP)

ROW FORMAT DELIMITED FIELDS TERMINATED BY '|'

STORED AS TEXTFILE

LOCATION '${stgLocation}';

Конфігураційний файл для інструменту Hive та HCatalog

<configuration>

<property>

<name>hive.metastore.cache.pinobjtypes</name>

<value>Table,Database,Type,FieldSchema,Order</value>

</property>

<property>

<name>javax.jdo.option.ConnectionDriverName</name>

<value>com.mysql.jdbc.Driver</value>

</property>

<property>

<name>javax.jdo.option.ConnectionUserName</name>

<value>hive</value>

</property>

<property>

<name>hive.auto.convert.join</name>

<value>true</value>

</property>

<property>

<name>fs.hdfs.impl.disable.cache</name>

<value>true</value>

</property>

<property>

<name>fs.file.impl.disable.cache</name>

<value>true</value>

</property>

<property>

<name>hive.metastore.warehouse.dir</name>

<value>/apps/hive/warehouse</value>

</property>

<property>

<name>hive.auto.convert.sortmerge.join</name>

<value>true</value>

</property>

<property>

<name>hive.metastore.client.socket.timeout</name>

<value>60</value>

</property>

<property>

<name>hive.optimize.bucketmapjoin</name>

<value>true</value>

</property>

<property>

<name>hive.optimize.bucketmapjoin.sortedmerge</name>

<value>true</value>

</property>

<property>

<name>hive.optimize.index.filter</name>

<value>true</value>

</property>

<property>

<name>hive.auto.convert.join.noconditionaltask.size</name>

<value>1000000000</value>

</property>

<property>

<name>hive.optimize.mapjoin.mapreduce</name>

<value>true</value>

</property>

<property>

<name>hive.auto.convert.join.noconditionaltask</name>

<value>true</value>

</property>

<property>

<name>hive.mapjoin.bucket.cache.size</name>

<value>10000</value>

</property>

<property>

<name>hive.auto.convert.sortmerge.join.noconditionaltask</name>

<value>true</value>

</property>

<property>

<name>hive.vectorized.execution.enabled</name>

<value>false</value>

</property>

<property>

<name>hive.security.authorization.enabled</name>

<value>false</value>

</property>

<property>

<name>hive.optimize.reducededuplication.min.reducer</name>

<value>1</value>

</property>

<property>

<name>hive.server2.enable.doAs</name>

<value>true</value>

</property>

<property>

<name>hive.mapred.reduce.tasks.speculative.execution</name>

<value>false</value>

</property>

<property>

<name>javax.jdo.option.ConnectionURL</name>

<value>jdbc:localhost/hive?createDatabaseIfNotExist=true</value>

</property>

<property>

<name>hive.enforce.bucketing</name>

<value>true</value>

</property>

<property>

<name>hive.metastore.execute.setugi</name>

<value>true</value>

</property>

<property>

<name>hive.enforce.sorting</name>

<value>true</value>

</property>

<property>

<name>hive.semantic.analyzer.factory.impl</name>

<value>org.apache.hivealog.cli.HCatSemanticAnalyzerFactory</value>

</property>

<property>

<name>hive.security.authorization.manager</name>

<value>org.apache.hadoop.hive.ql.security.authorization.DefaultHiveAuthorizationProvider</value>

</property>

<property>

<name>hive.map.aggr</name>

<value>true</value>

</property>

<property>

<name>hive.optimize.reducededuplication</name>

<value>true</value>

</property>

<property>

<name>hive.metastore.uris</name>

<value>thrift://localhost:9083</value>

</property>

<property>

<name>hive.exec.dynamic.partition</name>

<value>true</value>

</property>

<property>

<name>hive.exec.dynamic.partition.mode</name>

<value>nonstrict</value>

</property>

</configuration>