Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інфокомунікацій
(повна назва)

Кафедра Інформаційно-мережної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіті другий (магістерський)

Дослідження обчислювальної системи на базі Apache Hive

(тема)

Виконав:
здобувач 2 року навчання,
групи ІМІм-23-2
Борисов Я.С.
(прізвище, ініціали)

Спеціальність 172 Електронні комунікації та радіотехніка
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Інформаційно-мережна інженерія
(повна назва освітньої програми)

Керівник: доц. Омельченко С.В.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Безрук В.М.
(підпис) (прізвище, ініціали)

2025 р.

Не містить відомостей, заборонених до відкритого публікування

Студент _____ / Борисов Я.С./
(підпис) (прізвище та ініціали)

Керівник _____ / Омельченко С.В. /
(підпис) (прізвище та ініціали)

Харківський національний університет радіоелектроніки

Факультет Інфокомунікацій

Кафедра Інформаційно-мережної інженерії

Рівень вищої освіти другий (магістерський)

Спеціальність 172 Електронні комунікації та радіотехніка

(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформаційно-мережна інженерія

(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
« 28 » жовтня 2024 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Борисову Ярославу Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження обчислювальної системи на базі Apache Hive

затверджена наказом по університету від« 28» жовтня 2024 р. № 1148 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 23 січня 2025 р.

3. Вхідні дані до роботи Дослідити архітектуру та принципи роботи платформи Apache Hive. Розглянути процес розгортання Apache Hive. Дослідити специфіку виконання sql-запитів до датасету з Apache Hive. Дослідити шляхи оптимізації продуктивності кластеру Apache Hive .

4. Перелік питань, що потрібно опрацювати у роботі Вступ

1 Архітектура та принципи роботи Apache Hive

2. Розгортання Apache Hive

3. Виконання sql-запитів до датасету з Apache Hive

4. Оптимізація функціонування Apache Hive

Висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) слайди презентації в форматі Power Point (Назва та мета роботи, роль та особливості Apache Hive для кластерних обчислень, розгортання Apache Hive у кластері, виконання SQL-запитів, оптимізація кластерних обчислень з Apache Hive, висновки)

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Вступ	02.11.2024	виконано
2	Архітектура та принципи роботи Apache Hive	15.11.2024	виконано
3	Розгортання Apache Hive	05.12.2024	виконано
4	Виконання sql-запитів до датасету з Apache Hive	16.12.2024	виконано
5	Оптимізація функціонування Apache Hive	25.12.2024	виконано
6	Висновки	03.12.2025	виконано
7	Оформлення пояснювальної записки	07.01.2025	виконано

Дата видачі завдання 28 жовтня 2024 р.

Здобувач _____
(підпис)

Керівник роботи _____ доц. Омельченко С.В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 65 с., 26 рис., 15 джерел, 2 додатка.

Об'єкт дослідження – платформа Hive у середовищі Hadoop.

Мета роботи – дослідження функціонування та можливостей обробки Big Data з платформою Apache Hive.

Розглядається архітектура та принципи роботи інструментарію Apache Hive. Досліджуються процеси розгортання та конфігурування даної платформи на вузлі обчислювального кластеру. Вивчаються закономірності та специфіка роботи з Big Data.

Розглядаються поширені підходи до оптимізації обробки Великих Даних на базі Apache Hive у середовищі Hadoop.

APACHE HIVE, HADOOP, BIG DATA, MAPREDUCE ОБЧИСЛЕННЯ, ПАРТИЦІОНУВАННЯ ДАНИХ, ОБЧИСЛЮВАЛЬНИЙ КЛАСТЕР.

THE ABSTRACT

Explanatory note: 65 p., 26 fig., 15 sources, 2 app.

The object of research is the Hive platform in the Hadoop environment.

The aim of this paper is to study the functioning and processing capabilities of Big Data with the Apache Hive platform.

The architecture and operating principles of the Apache Hive toolkit are considered. The process of deploying and configuring a given platform on a computing cluster node is studied. The regularities and specifics of working with Big Data are studied. Common approaches to optimizing the processing of large data based on Apache Hive in the Hadoop environment are considered.

APACHE HIVE, HADOOP, BIG DATA, MAPREDUCE COMPUTING, DATA PARTITIONING, COMPUTING CLUSTER.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

SQL – (Structured Query Language) – структурована мова запитів, створена для того, щоб отримувати з бази даних необхідну інформацію;

СУБД – система управління базами даних;

HDFS – (Hadoop Distributed File System) – розподілена файлова система, призначена для зберігання та обробки великих об'ємів даних на кластерах обчислювальних вузлів;

API – (Application Programming Interface) інтерфейс прикладного програмування.

ВСТУП

Однією з характерних ознак перебігу процесу формування інформаційного суспільства є постійне та суттєве зростання обсягів інформації, яка міститься у файлосховищах та циркулює мережевими каналами [1].

Це, з одного боку, може сприйматися як негативний фактор, що вимагає:

– збільшення бітових швидкостей каналів;

– розширення ємностей файлосховищ.

Інакше кажучи, у таких умовах термін Великі Дані (Big Data) стає звичним, а самі Великі Дані перетворюються на повсякдення явище.

З іншого боку, формування великих обсягів даних про події, явища та перебіг численних процесів тощо дає змогу їх аналізувати та знаходити закономірності у даних, що дозволяє:

– виконувати прогнози перебігу явищ та процесів;

– прогнозувати ймовірність виникнення того чи іншого явища чи події;

– виконувати інтерполювання або екстраполювання даних числових рядів і т.д.

Аналітика Великих Даних знайшла своє застосування в таких галузях, як економіка, медицина, суспільствознавство, комп’ютерні науки, астрономія, кліматологія тощо [1,2].

При цьому, перші датасети можливо було обробляти у рамках єдиного обчислювального вузла. Сьогодні ж аналіз Big Data вимагає застосування набагато суттєвіших обчислювальних потужностей, де одним з підходів є перехід до кластерної архітектури обчислень. У такому випадку робота з даними виконується на кількох синхронизованих вузлах паралельно, а один з вузлів бере на себе роль керуючого. Зараз у ролі середовищ для кластерних обчислень широко застосовуються екосистеми Apache Spark та Apache Hadoop, які дозволяють досягати значних обчислювальних потужностей.

Разом з тим, для ряду задач використання кластерного підходу у чистому вигляді є недостатнім – наприклад, у випадку, коли датасети являють собою достатньо структуровані дані з великою кількістю полів (типовим представником таких датасетів є таблиці). У цьому разі класичний підхід передбачає обробку датасету цілком, що не є раціональним. Це пояснюється тим,

що за умови оптимізації параметрів кластеру збільшити продуктивність обробки можна лише додаванням нових вузлів та збільшення потужності існуючих [1,3].

Разом з тим, у якості альтернативи доцільно розглядати датасет як базу даних, яка дозволяє отримувати доступ до будь-яких записів та полів без необхідності обробки повного масиву даних, що суттєво прискорює процес обчислень.

Одним з інструментів, яких дозволяє реалізувати означений підхід, є платформа Hive у середовищі Hadoop. Це свідчить про актуальність теми дослідження.

1 АРХІТЕКТУРА ТА ПРИНЦИПИ РОБОТИ APACHE HIVE

1.1 Основні елементи Apache Hive

Apache Hive – це інструментарій у рамках Apache Hadoop, який дозволяє виконувати зберігання та обробку великих даних у розподіленому кластері з

а У сутності, Hive являє собою своєрідний препроцесор, який виконує

перетворення SQL-запитів на MapReduce-завдання над файлів у системі HDFS.

н Слід зазначити, що дані таблиць зберігаються у HDFS розподілено, як

файли різних форматів (за замовчуванням це CSV).

ч Для зберігання метаданих Hive використовує звичайну SQL базу даних -

найбільш поширена збірка в комплекті з Apache Derby (реляційна база даних, написана на Java).

о Разом з тим, доцільним буде зазначити, що у рамках Hive можуть не менш

ефективно використовуватися інші БД – такі, як MySQL, PostgreSQL або Oracle.

П

1. HCatalog – це компонента Hive, який відповідає за керування таблицями

та сховищами. Цей компонент також забезпечує користувачів різними

інструментами обробки Big Data (включаючи MapReduce та Pig) для більш

простого читання та запису даних.

2. WebHCat являє собою компоненту, яка дає змогу виконувати базові

операції (читання, запис та видалення) з метаданими у межах мережі Hive за

допомогою використання інтерфейсу HTTP (Hyper Text Transfer Protocol).

Рисунок 1.1 – Компоненти Hive

Окрім цього, WebHCat також використовується як сервер для розгортання середовища Hive.

1.2 Основні переваги Apache Hive для Big Data

В
и – індексація даних - Hive використовує індекси для швидшої роботи з
даними;
о – зберігання метаданих у СУБД сприяє значному зниженню часу
ремантичних перевірок на відповідність запиту конструкцій мови під час її
виконання;
с – зберігання даних на всіх вузлах кластера знижує ризик втрати інформації;
т – підтримка операцій із стислими даними, що зберігаються в екосистемі
Hadoop;
н – підтримка клієнта, встановленого на стороні користувача. Цей клієнт
взаємодіє із сервісами Hive, запущеними на сервері. Такий підхід добре
застосовується для додатків, написаних різними мовами програмування,
зокрема. JAVA, C++, Python, R. Використовуючи ці мови, користувач має
можливість підключення до інтерфейсу Hive для використання різних SQL-
команд та доступу до баз даних;
а – SQL-подібний синтаксис на діалекті HQL знижує поріг входу в
технологію. Такі SQL-подібні запити можуть вільно перетворюватися на
завдання MapReduce або Spark, що позбавляє користувача зайвих дій з
перетворення даних Hive.

Інструментарій Apache Hive було створено для реалізації технології Hadoop MapReduce у форматі, що дозволяє вести розподілену обробку даних.
і Разом з тим, необхідно було вирішити ще одну гостру проблему, а саме -
неможливість швидкого навчання фахівців та аналітиків навичкам використання
Hadoop. Щоб цього уникнути, розробниками було вирішено у якості інтерфейсу далі використовувати SQL-подібний інтерфейс, так як він широко використовується фахівцями в галузі роботи з даними. При цьому, хоча SQL міг задовольнити більшість вимог аналітики, розробники Apache Hive вирішили реалізувати можливості програмування Hadoop для проектування розподілених
д
о Отже, Hive з'явився як синтез двох ідей:
д
т
я

– декларативна мова на основі SQL, для того, щоб аналітики та дата-інженери могли достатньо легко створювати власні сценарії роботи з великими даними;

– можливості підключення різних мов програмування (Python, JAVA, Scala, C++), які використовують SQL інтерфейс для розробки спеціалізованих додатків.

Таким чином, завдяки розподіленій роботі з даними та SQL-інтерфейсу, Apache Hive є дуже зручним та надійним засобом для роботи з великими даними. Саме тому Hive є невід'ємною частиною екосистеми Hadoop, що є універсальним рішенням для обробки Big Data.

З іншого боку, більшість класичних СУБД, таких як Postgres, MySQL або Oracle не мають такої гнучкості в масштабуванні при обробці великих масивів даних і при досягненні більшого обсягу подальша підтримка стає проблематичною.

Власне, Hive об'єднує дві переваги [4]:

– масштабованість MapReduce

– зручність використання SQL для вибірок із даних.

1.3 Загальний опис платформи

Рисунок 1.2 демонструє ключову архітектуру платформи.

Отже, як вже зазначалося раніше, Hive являє собою своєрідний двигун, який перетворює SQL-запити в ланцюжки map-reduce завдань.

Даний двигун включає такі компоненти, як [3-5]:

– Parser (розбирає вхідні SQL-запити);

– Optimimer (оптимізує запит для досягнення більшої ефективності);

– Planner (планує завдання на виконання);

– Executor (запускає завдання на фреймворку MapReduce.

Для роботи Hive також необхідне сховище метаданих. Справа в тому, що SQL передбачає роботу з такими об'єктами як база даних, таблиця, колонки, рядки, осередки і т.д.

Оскільки самі дані, які використовує Hive, зберігаються просто у вигляді файлів на hdfs — необхідно десь зберігати відповідність між об'єктами hive і реальними файлами.

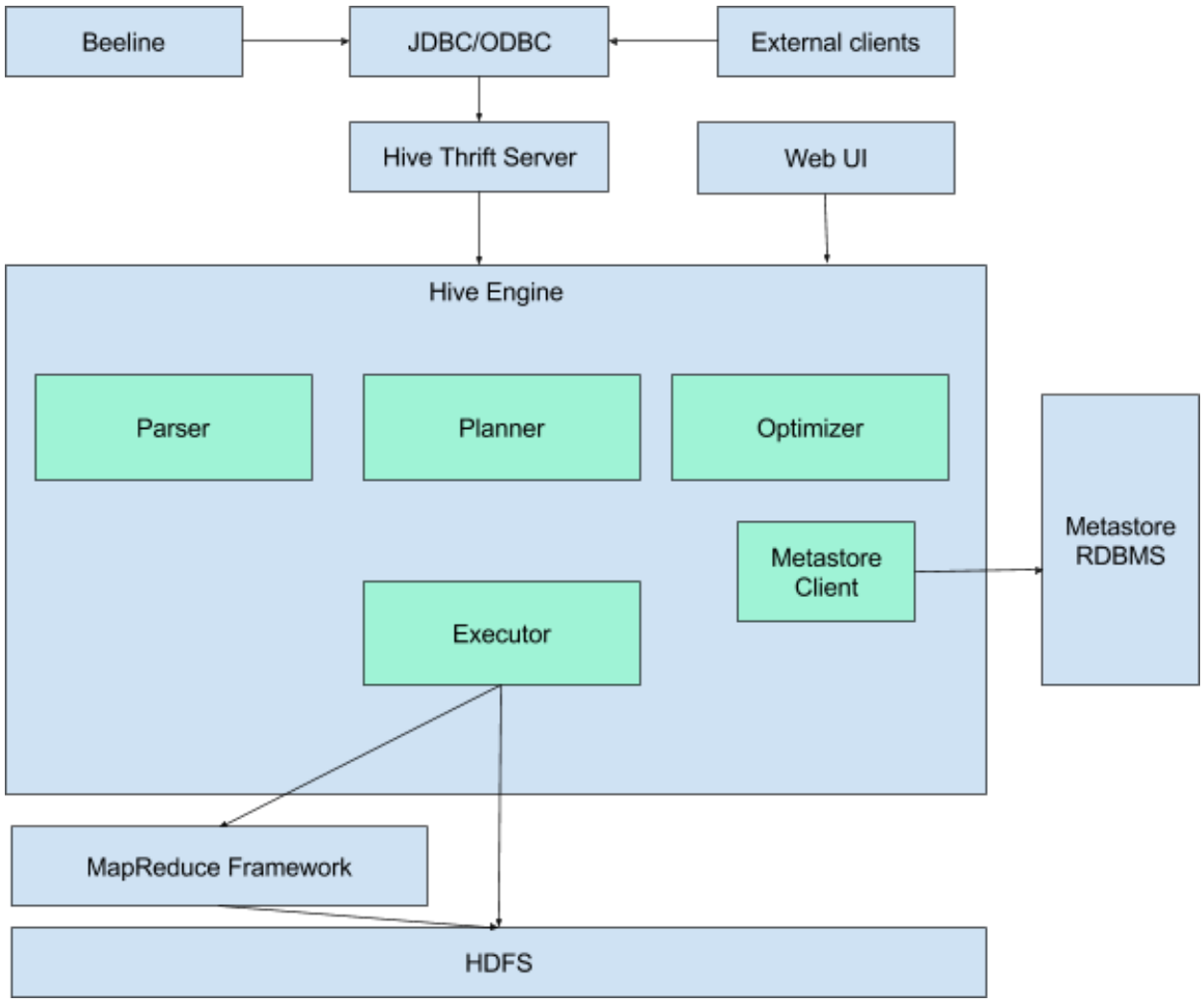


Рисунок 1.2 – Архітектура Hive

Як metastorage використовується звичайна реляційна СУБД, така як MySQL, PostgreSQL чи Oracle.

1.4 Використання Command line interface

Для того щоб розпочати роботу з Hive, найпростіше скористатися його командним рядком. Сучасна утиліта для роботи з hive називається beeline. Для цього на будь-якій машині в hadoop-кластері зі встановленим hive достатньо виконати команду [3, 6]:

```
$beeline
```

Далі необхідно встановити з'єднання з hive-сервером:

```
beeline> !connect jdbc:hive2://localhost:10000/default root root
Connecting to jdbc:hive2://localhost:10000/default
Connected to: Apache Hive (version 1.1.0-cdh5.7.0)
Driver: Hive JDBC (version 1.1.0-cdh5.7.0)
Transaction isolation: TRANSACTION_REPEATABLE_READ
0: jdbc:hive2://localhost:10000/default>
```

де root root — це ім'я користувача та пароль. Після цього ми отримуємо у розпорядження командний рядок, у якому можна вводити команди hive.

Також іноді буває зручно не вводити SQL-запити в командний рядок beeline, а попередньо зберегти та редагувати їх у файлі, а потім виконати всі запити з файлу. Для цього потрібно виконати beeline з параметрами підключення до бази даних та параметром -f, що вказує ім'я файлу, що містить запити [3, 6]:

```
beeline -u jdbc:hive2://localhost:10000/default -n root -p root -f sorted.sql
```

1.5 Об'єкти Hive

При роботі з Hive можна виділити такі об'єкти, якими оперує платформа:

- база даних;
- таблиця;
- партія (partition);
- бакет (bucket).

Розглянемо кожен із них докладніше.

1.5.1 База даних

База даних є аналогом бази даних у реляційних СУБД. База даних є простором імен, що містить таблиці. Команда створення нової бази даних виглядає так [3-6]:

```
CREATE DATABASE|SCHEMA [IF NOT EXISTS] <database name>
```

Database і Schema у даному контексті це синоніми. Необов'язкова директива IF NOT EXISTS створює базу даних тільки в тому випадку, якщо вона ще не існує.

Приклад створення бази даних:

```
CREATE DATABASE userdb;
```

Для перемикання на відповідну базу даних може бути використано команду USE:

```
USE userdb;
```

1.5.2 Таблиця

Таблиця в Hive представляє собою аналог таблиці в класичній реляційній БД. Основна відмінність - що дані hive-таблиць зберігаються просто у вигляді звичайних файлів на HDFS [3-6].

Наприклад, це можуть бути звичайні текстові csv-файли, бінарні sequence-файли, складніші колонкові parquet-файли та інші формати. Але в будь-якому випадку дані, відносно яких налаштовано hive-таблиця, дуже легко прочитати і не з Hive.

Таблиці у hive бувають двох типів. Перший – це класична таблиця, дані до якої додаються за допомогою hive.

Розглянемо приклад створення такої таблиці:

```
CREATE TABLE IF NOT EXISTS employee ( eid int, name String,
salary String, destination String)
COMMENT 'Employee details'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;
```

Лістинг вище створює таблицю, дані у якій зберігатимуться як у звичайних csv-файлах, колонки у яких розділені символом табуляції. Після цього дані до таблиці можна завантажити.

Припустимо, що у домашній директорії на hdfs існує файл sample.txt вигляду:

```
1201    Gopal  45000  Technical manager
1202    Manisha      45000  Proof reader
1203    Masthanvali  40000  Technical writer
1204    Kiran  40000  Hr Admin
1205    Kranthi      30000  Op Admin
```

Для завантаження даних необхідно виконати наступну команду:

```
LOAD DATA INPATH '/user/root/sample.txt'
OVERWRITE INTO TABLE employee;
```

У результаті даної команди hive перемістить дані, що зберігається в зазначеному файлі, до сховища hive. Переконатися в цьому можна, прочитавши дані безпосередньо з файлу в сховище hive в HDFS:

```
[root@quickstart ~]# hadoop fs -text /user/hive/warehouse/userdb.db/employee/*
1201 Gopal 45000 Technical manager
1202 Manisha 45000 Proof reader
1203 Masthanvali 40000 Technical writer
1204 Kiran 40000 Hr Admin
1205 Kranthi 30000 Op Admin
```

Класичні таблиці можна також створювати як результат select-запиту до інших таблиць [4, 5]:

```
0: jdbc:hive2://localhost:10000/default> CREATE TABLE big_salary as SELECT * FROM employee WHERE salary > 40000;

0: jdbc:hive2://localhost:10000/default> SELECT * FROM big_salary;
+-----------------+------------------+------------ --------+--
----------------------+--+
| big_salary.eid | big_salary.name | big_salary.salary | big_salary.destination |
```

```
    +----------------+-----------------+----------- --------+--
----------------------+--+
    | 1201 | Gopal | 45000 | Technical manager |
    | 1202 | Manisha | 45000 | Proof reader |
    +----------------+-----------------+----------- --------+--
----------------------+--+
```

До речі, команда SELECT для створення таблиці у даному разі вже запустить mapreduce-задачу [5-8].

Інший тип таблиць - зовнішня таблиця, дані у яку завантажуються зовнішніми системами, без участі Hive.

Для роботи з зовнішніми таблицями попередньо потрібно вказати ключове слово EXTERNAL і вказати шлях до директорії, у яких зберігаються файли:

```
    CREATE EXTERNAL TABLE IF NOT EXISTS employee_external ( eid
int, name String,
    salary String, destination String)
    COMMENT 'Employee details'
    ROW FORMAT DELIMITED
    FIELDS TERMINATED BY '\t'
    LINES TERMINATED BY '\n'
    STORED AS TEXTFILE
    LOCATION '/user/root/external_files/';
```

Після цього таблицею можна користуватися так само як і звичайними таблицями Hive.

Зручним є те, що можливо просто скопіювати файл у потрібну директорію в hdfs, а Hive автоматично підхоплюватиме нові файли при запитах до відповідної таблиці. Це дуже зручно при роботі, наприклад, з логами.

1.5.3 Партія (partition)

Так як Hive являє собою двигун для трансляції SQL-запитів в mapreduce-завдання, то зазвичай навіть найпростіші запити до таблиці призводять до повного сканування даних у цій таблиці [5, 6].

Щоб уникнути повного сканування даних, з деяких колонок таблиці можна зробити поділ цієї таблиці. Це означає, що дані, що стосуються різних значень, будуть фізично зберігатися в різних папках HDFS.

Для створення партиціонованої таблиці необхідно вказати, за якими колонками буде зроблено партиціонування:

```
CREATE TABLE IF NOT EXISTS employee_partitioned ( eid int, name String,
salary String, destination String)
COMMENT 'Employee details'
PARTITIONED BY (birth_year int, birth_month string)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;
```

При завантаженні даних у таку таблицю необхідно явно вказати, до якої партиції виконується завантаження даних:

```
LOAD DATA INPATH '/user/root/sample.txt' OVERWRITE
INTO TABLE employee_partitioned
PARTITION (birth_year=1998, birth_month='May');
```

Подивимося, тепер як виглядає структура директорій:

```
[root@quickstart ~]# hadoop fs -ls /user/hive/warehouse/employee_partitioned/
Found 1 items
drwxrwxrwx - root supergroup 0 2016-05-08 15:03 /user/hive/warehouse/employee_partitioned/birth_year=1998
[root@quickstart ~]# hadoop fs -ls -R /user/hive/warehouse/employee_partitioned/
drwxrwxrwx - root supergroup 0 2016-05-08 15:03 /user/hive/warehouse/employee_partitioned/birth_year=1998
drwxrwxrwx - root supergroup 0 2016-05-08 15:03 /user/hive/warehouse/employee_partitioned/birth_year=1998/birth_month=May
-rwxrwxrwx 1 root supergroup 161 2016-05-08 15:03 /user/hive/warehouse/employee_partitioned/birth_year=1998/birth_month=May/sample.txt
```

Як видно з попереднього лістингу, структура директорій виглядає так, що кожній партиції відповідає окрема папка на hdfs. Тепер, якщо ми будемо запускати будь-які запити, вказавши в умові WHERE обмеження на значення партій - mapreduce візьме вхідні дані тільки з відповідних папок [7].

У разі External таблиць партиціонування працює аналогічним чином, але подібну структуру директорій доведеться створювати вручну.

Партіціонування дуже зручно наприклад для поділу логів за датами, оскільки зазвичай будь-які запити за статистикою містять обмеження за датами. Це дозволяє значно скоротити час запиту.

1.5.4 Бакет

Партиціонування допомагає скоротити час обробки, якщо зазвичай при запитах відомі обмеження на значення якогось стовпця. Однак воно не завжди застосовне. Наприклад, якщо кількість значень у стовпці дуже велика. Наприклад, це може бути ID користувача в системі, що містить кілька мільйонів користувачів [3-6].

У цьому випадку доцільно виконати поділ таблиці на бакети. До одного бакету потрапляють рядки таблиці, для яких значення збігається зі значенням хеш-функції, обчисленої для певної колонки.

При будь-якій роботі з бакетованими таблицями необхідно активувати підтримку бакетів в Hive (інакше hive працюватиме з ними як зі звичайними таблицями):

```
set hive.enforce.bucketing=true;
```

Для створення таблиці, розбитої на бакети, використовується конструкція

```
set hive.enforce.bucketing=true;

CREATE TABLE employee_bucketed ( eid int, name String, salary String, destination String)
CLUSTERED BY(eid) INTO 10 BUCKETS
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;
```

При цьому, так як команда Load використовується для простого переміщення даних у сховище Hive - в даному випадку для завантаження вона не підходить, так як дані необхідно обробити, правильно розбивши їх на бакети [7]. Таким чином, у цьому випадку дані потрібно завантажити за допомогою команди INSERT з іншої таблиці (наприклад із зовнішньої таблиці):

```
set hive.enforce.bucketing=true;
FROM employee_external INSERT OVERWRITE TABLE employee_bucketed SELECT *;
```

Після виконання команди необхідно переконатися, що дані справді поділилися на 10 частин:

```
[root@quickstart ~]# hadoop fs -ls /user/hive/warehouse/employee_bucketed
Found 10 items
-rwxrwxrwx 1 root supergroup 31555556 2016-05-08 16:04 /user/hive/warehouse/employee_bucketed/000000_0
-rwxrwxrwx 1 root supergroup 31555556 2016-05-08 16:04 /user/hive/warehouse/employee_bucketed/000001_0
-rwxrwxrwx 1 root supergroup 31555556 2016-05-08 16:04 /user/hive/warehouse/employee_bucketed/000002_0
-rwxrwxrwx 1 root supergroup 31555556 2016-05-08 16:04 /user/hive/warehouse/employee_bucketed/000003_0
-rwxrwxrwx 1 root supergroup 31555556 2016-05-08 16:04 /user/hive/warehouse/employee_bucketed/000004_0
-rwxrwxrwx 1 root supergroup 31555556 2016-05-08 16:04 /user/hive/warehouse/employee_bucketed/000005_0
-rwxrwxrwx 1 root supergroup 31555556 2016-05-08 16:04 /user/hive/warehouse/employee_bucketed/000006_0
-rwxrwxrwx 1 root supergroup 31555556 2016-05-08 16:04 /user/hive/warehouse/employee_bucketed/000007_0
-rwxrwxrwx 1 root supergroup 31555556 2016-05-08 16:04 /user/hive/warehouse/employee_bucketed/000008_0
```

Отже, виконуючи запити за даними, що належать до певного користувача, нам не потрібно виконувати сканування усієї таблиці, а лише 1/10 частину цієї таблиці.

2 РОЗГОРТАННЯ APACHE HIVE

Платформа Hive являє собою своєрідну надбудову над Hadoop, відтак у кластері спочатку розгорнути дану систему.

За умови, що Hadoop попередньо встановлено та налаштовано, виконується інсталяція Hive. При цьому, різниці для однонодової та багатонодової конфігурацій кластеру не існує, оскільки фреймворк встановлюється лише на Майстер-вузлі (рис.2.1) [3, 5, 8].

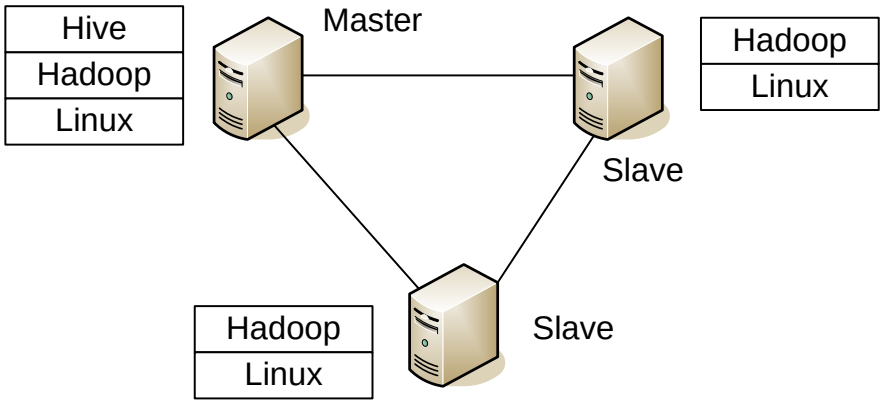


Рисунок 2.1 – Приклад програмної структури обчислювального кластеру з трьома нодами на базі Hadoop

2.1 Встановлення Hive

Для початку необхідно запустити Hadoop на Майстер-вузлі після чого завантажити необхідні дистрибутиви Hive та компоненти Derby.

При цьому, необхідно забезпечити сумісність версій Hive, Hadoop, Derby та Java, для чого доцільно скористатися відповідними відомостями з сайту розробника [9].

Після цього виконується розпакування завантажених архівів у директорію /usr/local/, для цього виконуємо ряд команд [8]:

```
cd home/user/Downloads
```

де user – поточний користувач, під яким завантажили файл,

```
tar zxvf apache-hive-2.3.6-bin.tar.gz
```

Далі переносимо вміст:

```
sudo mv apache-hive-2.3.6-bin /usr/local/hive
```

Звертаємося до створеного каталогу:

```
cd /usr/local/hive
ls
```

Після цього можемо переглянути вміст директорії, як показано на рисунку 2.2.

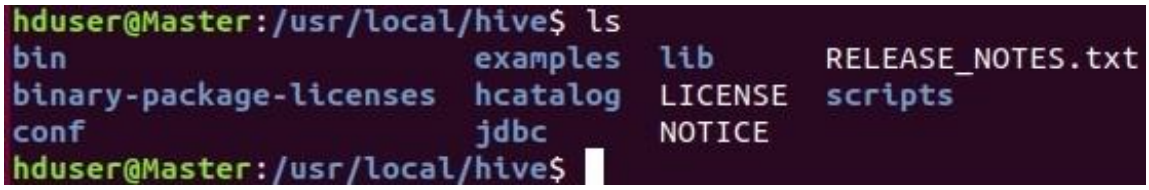


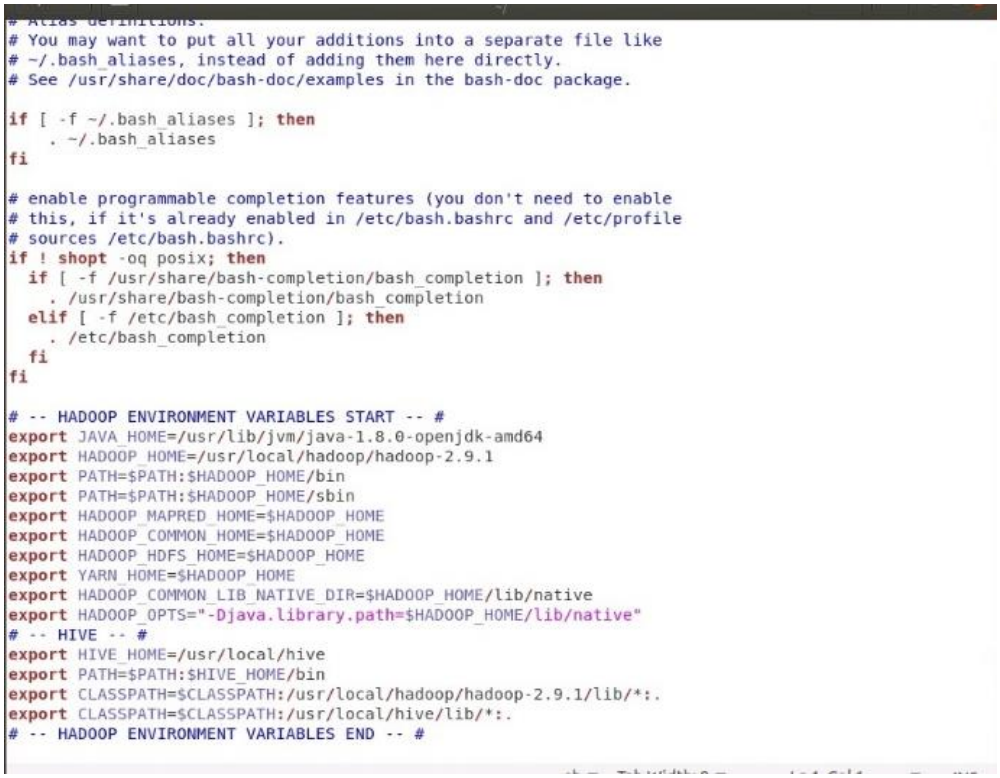
Рисунок 2.2 – Вміст директорії hive

Далі необхідно прописати змінні оточення у файлі bashrc, для чого потрібно перейти до домашнього каталогу та відкрити файл у текстовому редакторі.

У кінці даного

```
export HIVE_HOME=/usr/local/hive
export PATH=$PATH:$HIVE_HOME/bin
export CLASSPATH=$CLASSPATH:/usr/local/hadoop/hadoop-2.9.1/lib/*:.
export CLASSPATH=$CLASSPATH:/usr/local/hive/lib/*:.
```

У нашому випадку маємо наступне (рис.2.3).



```
# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi

# -- HADOOP ENVIRONMENT VARIABLES START -- #
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-amd64
export HADOOP_HOME=/usr/local/hadoop/hadoop-2.9.1
export PATH=$PATH:$HADOOP_HOME/bin
export PATH=$PATH:$HADOOP_HOME/sbin
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib/native"
# -- HIVE -- #
export HIVE_HOME=/usr/local/hive
export PATH=$PATH:$HIVE_HOME/bin
export CLASSPATH=$CLASSPATH:/usr/local/hadoop/hadoop-2.9.1/lib/*:.
export CLASSPATH=$CLASSPATH:/usr/local/hive/lib/*:.
# -- HADOOP ENVIRONMENT VARIABLES END -- #
```

Рисунок 2.3 – Внесення змінних оточення у файл bashrc

Далі задаємо каталог

```
cd $HIVE_HOME/conf
sudo cp hive-env.sh.template hive-env.sh
```

У даному файлі вкажемо шлях до Hadoop:

```
sudo gedit hive-env.sh
```

Врешті-решт, додамо рядок:

```
export HADOOP_HOME=/usr/local/hadoop/hadoop-2.9.1
```

2.2 Встановлення бази даних Derby

Попередньо завантажений архів db-derby-10.14.2.0-bin з дистрибутивом розміщуємо у каталог /usr/local/Derby, після розпакування та інсталяції у файл bash додаємо наступні рядки:

```
export DERBY_HOME=/usr/local/Derby
export PATH=$PATH:$DERBY_HOME/bin
export CLASSPATH=$CLASSPATH:$DERBY_HOME/lib/derby.jar:$DERBY_HOME/lib/derbytools.jar
```

Далі створюємо файл мета-сховища Derby, для чого виконуємо команду:

```
sudo mkdir $DERBY_HOME/data
```

2.3 Налаштування мета-сховища та запуск Hive

У даному випадку використовується основний файл конфігурації Hive, а

```
cd $HIVE_HOME/conf
cp hive-default.xml.template hive-site.xml
```

Даний файл конфігурується, при цьому у нього вносяться наступні рядки, як показано рис.2.4.

```
<property>
   <name>system:java.io.tmpdir</name>
   <value>/tmp/hive/java</value>
 </property>
 <property>
   <name>system:user.name</name>
   <value>${user.name}</value>
 </property>
<property>
       <name>hive.metastore.schema.verification</name>
       <value>false</value>
</property>
<property>
   <name>javax.jdo.option.ConnectionURL</name>
   <value>jdbc:derby://localhost:1527/usr/local/hive/metastore_db;create=true </value>
   <description>JDBC connect string for a JDBC metastore </description>
</property>
```

Рисунок 2.4 – Конфігурування файлу hive-site.xml

hive-site.xml

Тут рядок jdbc:derby, має відповідати реальному шляху до метасховища.

Далі потрібно створити файл jpox.properties у директорії $HIVE_HOME/conf.

У файлі вносити нас тупні зміст (р

```
javax.jdo.PersistenceManagerFactoryClass =
org.jpox.PersistenceManagerFactoryImpl
org.jpox.autoCreateSchema = false
org.jpox.validateTables = false
org.jpox.validateColumns = false
org.jpox.validateConstraints = false
org.jpox.storeManagerType = rdbms
org.jpox.autoCreateSchema = true
org.jpox.autoStartMechanismMode = checked
org.jpox.transactionIsolation = read_committed
javax.jdo.option.DetachAllOnCommit = true
javax.jdo.option.NontransactionalRead = true
javax.jdo.option.ConnectionDriverName =
org.apache.derby.jdbc.ClientDriver
javax.jdo.option.ConnectionURL =
jdbc:derby://Master1:1527/usr/local/hive/metastore_db;create =
true
javax.jdo.option.ConnectionUserName = APP
javax.jdo.option.ConnectionPassword = mine
```

Рисунок 2.5 – Конфігураційний файл метасховища Hive

Після цього виконуємо ініціалізацію метасховища командами:

```
cd $HIVE_HOME/bin
schematool -initSchema -dbType derby
```

На наступному кроці здійснюється налаштування директорій HDFS для Hive та призначення їм права доступу:

```
hadoop fs -mkdir /tmp
hadoop fs -mkdir /user/hive/warehouse
hadoop fs -chmod g+w /tmp
hadoop fs -chmod g+w /user/hive/warehouse
```

У підсумку цього можемо виконати запуск Hive командами:

```
cd $HIVE_HOME/bin
hive
```

Ознакою коректного налаштування та запуску командного рядку Hive може біти лістинг, як показано рис.2.6.

Рисунок 2.6 – Звернення до командного рядку Hive

2.4 Налаштування мета-сховища MySQL для Hive

Якщо компонента Derby використовується у тестових збірках, то у випадку реальних кластерів використовується метасховище MySQL.

2.4.1 Перевірка наявності СУБД MySQL у системі

Перевірка виконується за допомогою команди mysql. На серверних системах Linux MySQL зазвичай у комплекті за замовчуванням, тому для майстер-вузлів кластеру вигідно використовувати їх. В іншому випадку необхідно виконати інсталяцію, скориставшись командами [11]:

```
sudo apt-get install mysql-server
```

Після успішної установки входимо у консоль mysql:

```
sudo mysql -u root -p
```

2.4.2 Встановлення mysql java конектор

Тут використовується команда:

```
sudo apt-get install libmysql-java
```

Н

```
ln -s /usr/share/java/mysql-connector-java.jar $HIVE_HOME/lib/mysql-connector-java.jar
```

2.4.3 Налаштування метасховища Hive та створення користувача

Д

```
sudo mysql -u root –p
```

Після вводу паролю вхід буде здійснено. Для створення бази для метасховища виконуємо команди:

```
mysql> CREATE DATABASE metastore;
mysql> USE metastore;
```

Далі цього створимо користувача для бази даних. Якщо при створенні користувача система видає сповіщення про недостатню стійкість пароля, то треба встановити складніший, або змінити змінні mysql типу validate_password%:

```
mysql> CREATE USER 'hiveuser'@'%' IDENTIFIED BY 'hivepassword';
mysql> GRANT all on *.* to 'hiveuser'@localhost identified by 'hivepassword';
mysql> flush privileges;
```

Після цього слід виконати відповідні налаштування Hive. Для цього потрібно внести зміни до файлу hive-site.xml (файл створюється з шаблону hive-default.xml.template):

```
sudo gedit $HIVE_HOME/conf/hive-site.xml
```

Тут слід зазначити, що усі необхідні властивості вже прописані у файлі, тому необхідно лише змінити їх значення на потрібні у конкретній ситуації (рис.2.7).

```
<property>
  <name>javax.jdo.option.ConnectionURL</name>
  <value>jdbc:mysql://localhost/metastore?createDatabaseIfNotExist=true&amp;useSSL=false</value>
  <description>JDBC connect string for a JDBC metastore </description>
</property>
<property>
   <name>javax.jdo.option.ConnectionUserName</name>
   <value>hiveuser</value>
   <description>Username to use against metastore database</description>
</property>
<property>
   <name>javax.jdo.option.ConnectionPassword</name>
   <value>hivepassword</value>
   <description>password to use against metastore database</description>
</property>
<property>
   <name>javax.jdo.option.ConnectionDriverName</name>
   <value>com.mysql.jdbc.Driver</value>
   <description>Driver class name for a JDBC metastore</description>
</property>
<property>
   <name>hive.metastore.port</name>
   <value>10802</value>
   <description>Hive metastore listener port</description>
</property>
```

Рисунок 2.7 – Критичні параметри файлу hive-site.xml, які потребують корегування

Далі у початок файлу вносяться додаткові налаштування каталогів, як показано на рисунку 2.8.

```
<property>
    <name>system:java.io.tmpdir</name>
    <value>/tmp/hive/java</value>
</property>
<property>
    <name>system:user.name</name>
    <value>${user.name}</value>
</property>
```

Рисунок 2.8 – Додаткові налаштування каталогів файлу hive-site.xml

Тепер залишилося лише застосувати схему для метасховища. Для цього виконуються наступні команди:

```
cd $HIVE_HOME/bin/
schematool -dbType mysql -initSchema
```

Після цього можемо виконати запуск Hive та тестування середовища, запустивши команди створення таблиць.

2.5 Налаштування клієнт-серверної взаємодії Hive (HiveServer2 та beeline)

За замовчуванням робота з таблицями hive локально виконується за допомогою командного рядка Hive. Разом з тим, за допомогою вбудованого клієнту beeline може бути налагоджено підключення до віддаленого серверу hive (HiveServer2).

Дослідимо процес налаштування та запуску серверу HiveServer2 з наступним підключенням до нього.

До даного файлу додаються змінні hadoop.proxyuser.hduser.hosts та hadoop.proxyuser.hduser.groups зі значенням *, як показано на рисунку 2.9.

```
    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>
<property>
<name>fs.defaultFS</name>
<value>hdfs://Master:9000</value>
</property>
<property>
<name>hadoop.temp.dir</name>
<value>/usr/local/hadoop/hadoop_tmp</value>
</property>

<property>
<name>hadoop.proxyuser.hduser.hosts</name>
<value>*</value>
</property>

<property>
<name>hadoop.proxyuser.hduser.groups</name>
<value>*</value>
</property>
</configuration>
```

Рисунок 2.9 – Додавання змінних hadoop.proxyuser.hduser.hosts та hadoop.proxyuser.hduser.groups

У першу чергу до файлу налаштувань Hadoop необхідно додати змінні для доступу користувача до Hive. Спершу необхідно відкрити для редагування файл core-site.xml директорії hadoop. У нашому випадку команда така:

```
sudo gedit $HADOOP_HOME/etc/hadoop/hadoop-2.9.1/core-site.xml
```

Далі необхідно внести зміни до файлу hive-site.xml. У даному файлі необхідно вказати номер порту thrift-сервера, рівень логування помилок сервером, а також режим роботи сервера (http або binary). Налаштування, використані у нашому випадку представлені на рис.2.10.

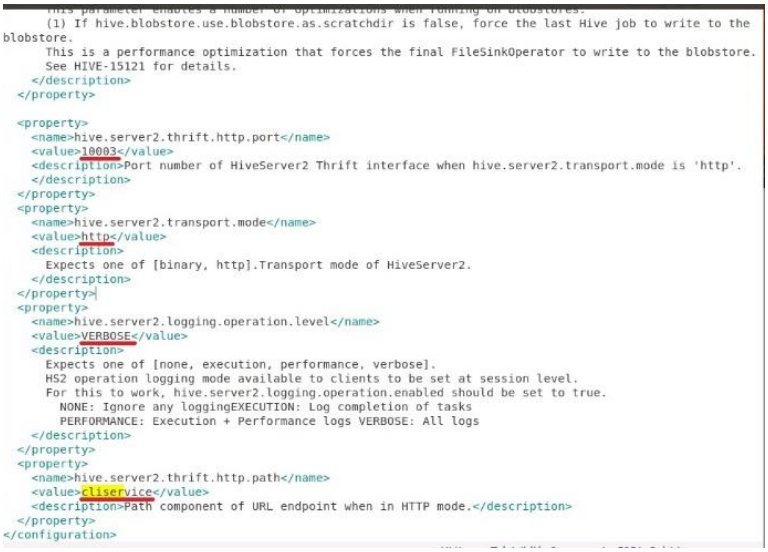


Рисунок 2.10 – Зміни, які внесено до файлу hive-site.xml

Розглянемо дані налаштування докладніше:

- hive.server2.thrift.http.port - важливий параметр, за номером цього порту виконується звернення до серверу. Значення може бути довільним, головне, щоб порт був не зайнятий. Якщо при спробі підключення буде видаватися помилка connection refused, необхідно змінити номер порту;

- hive.server2.transport.mode - важливий параметр, режим, в якому здійснюватиметься взаємодія, у нашому випадку вибрано http.

Інші параметри можна не змінювати, тоді як параметр cliservice взагалі не с

л У свою чергу, високий рівень логування помилок дозволить отримати

б більше інформації про виникаючі інциденти.

д

з

На цьому кроці все готове до запуску сервера. Після запуску Hadoop сервер що

```
cd $HIVE_HOME/bin
hive --service hiveserver2
```

може бути запущено командою:

Результат виконання команди показано на рисунку 2.11.

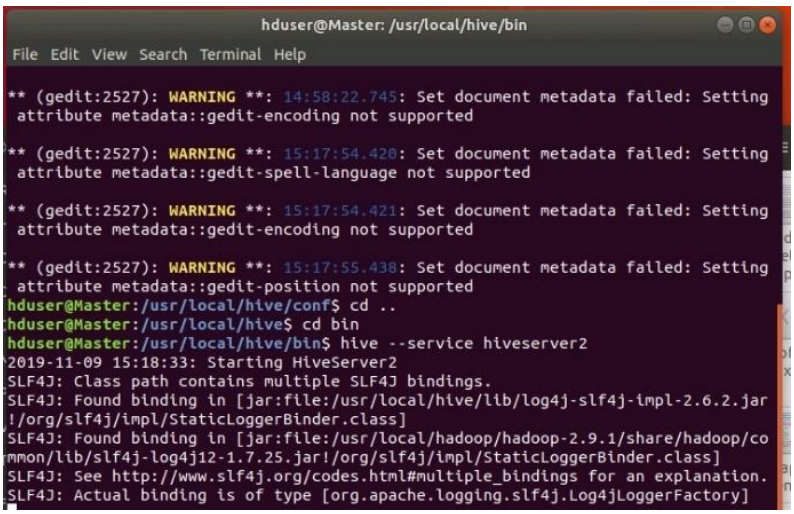


Рисунок 2.11 – Запуск серверу HiveServer2

Як бачимо, сервер блокує командний рядок, оскільки у ньому будуть відображатися результати роботи за сервера, тому для подальшої роботи необхідно відкрити ще один термінал (рис.2.12).

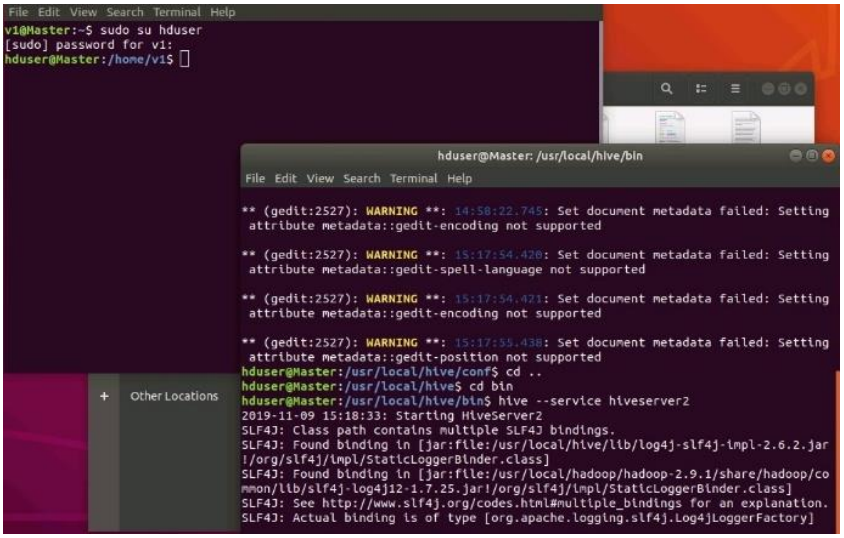


Рисунок 2.12 – Запуск додаткового терміналу

Після активації додаткового терміналу виконаємо запуск вбудованого hive-клієнту beeline.

При цьому, сам Клієнт beeline знаходиться у директорії $HIVE_HOME/bin. Для його запуску достстньо скористатися командою beeline (рис.2.13).

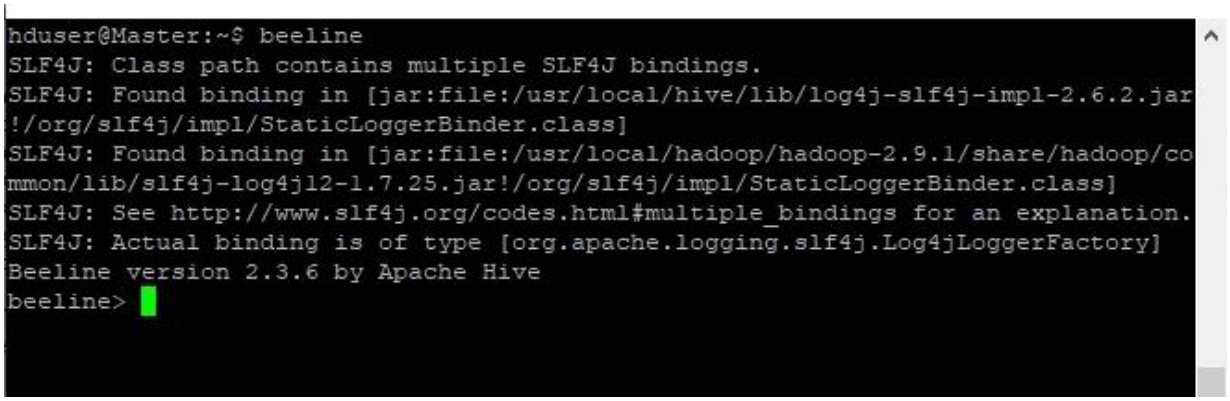


Рисунок 2.13 – Запуск клієнту beeline

Таким чином, виконано вхід до командного рядка клієнту beeline, де виконуються команди підключення до віддаленої бази та команди роботи з базами та таблицями.

Далі виконаємо підключення до серверу за допомогою команди, показаної далі (рис.2.14):

```
!connect
jdbc:hive2://IP_адреса_сервера:10003/default;transportMode=http;ht
tpPath=cliservice;
```

Розглянемо параметри команд:

- IP_адреса_сервера - тут може бути використано також ім'я машини, до якої підключаємося, якщо є DNS або відповідний запис у файлі hosts;

- Порт (:10003) – параметр, вказаний у файлі hive-site.xml;

- transportMode=http;httpPath=cliservice; - ці параметри також встановлено у файлі hive-site.xml.

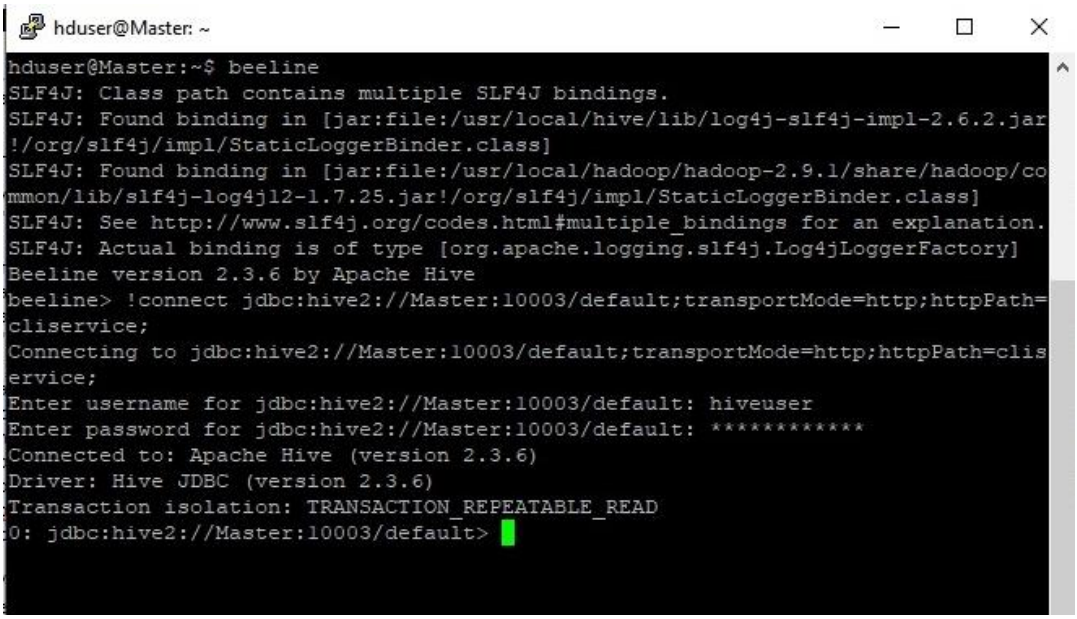


Рисунок 2.14 – Процес підключення до серверу за допомогою beeline

З рисунку 2.14 можна бачити, що підключення до бази даних було успішно здійснено. Для перевірки роботи далі можна виконати команду show tables; у відповідь отримаємо список

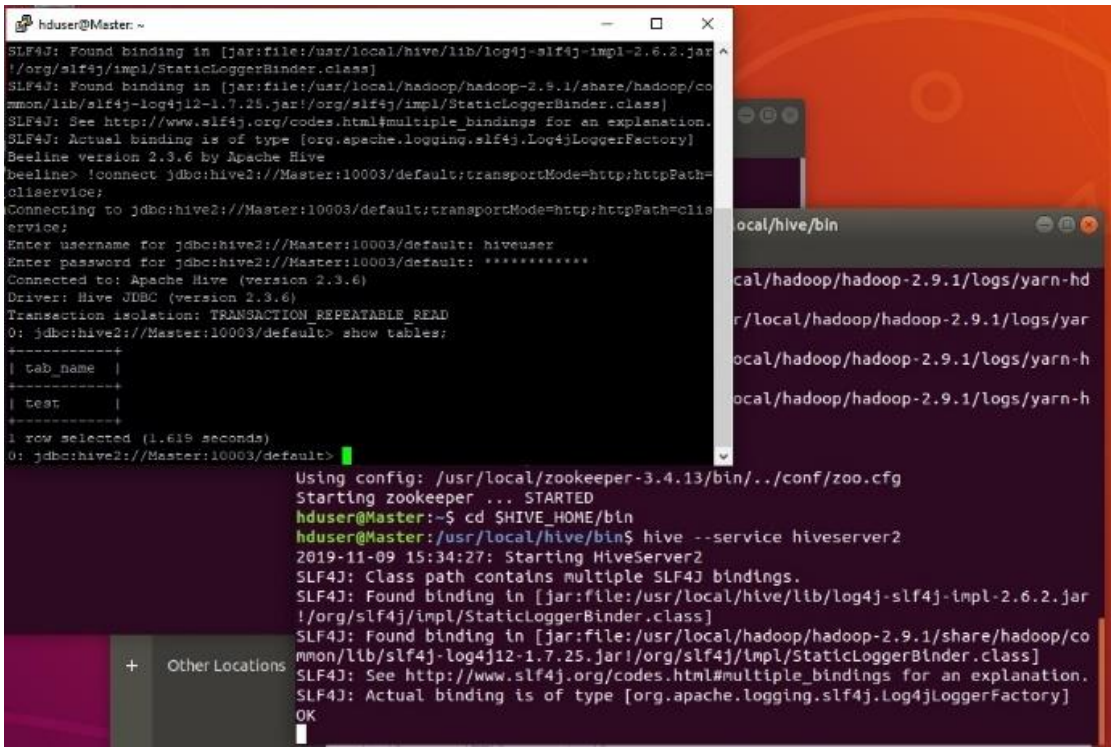


Рисунок 2.15 – Підключення до бази даних на HiveServer2 з переглядом таблиць

3 ВИКОНАННЯ SQL-ЗАПИТІВ ДО ДАТАСЕТУ З APACHE HIVE

3.1 Вихідні дані

Як зазначається у багатьох джерелах, одним із джерел великих даних є бази даних. Для того, щоб дослідити закономірності роботи з даним у Hive, використаємо датасет у вигляді CSV-файлу, де містяться дані від RSS-каналів. Розмір файлу зараз перевищує 150 Мб. Оскільки розмір блоку HDFS дорівнює 128 Мб, у цьому випадку вже маємо розподілене зберігання.

Саму структуру таблиці CSV-файлу для зберігання новин з RSS-каналів подано на рис.3.1.

#	Имя	Тип	Сравнение	Атрибуты	Null	По умолчанию	Дополнительно
1	id	int(10)			Нет	*Нет*	AUTO_INCREMENT
2	agency	varchar(20)	utf8_general_ci		Нет	*Нет*	
3	head	varchar(1000)	utf8_general_ci		Нет	*Нет*	
4	link	varchar(1000)	utf8_general_ci		Нет	*Нет*	
5	text	text	utf8_general_ci		Нет	*Нет*	
6	pubdate	varchar(50)	utf8_general_ci		Нет	*Нет*	
7	garbage	int(3)			Нет	*Нет*	

Рисунок 3.1 – Структура вихідного CSV-файлу

Розглянемо поля даної таблиці:

- agency - текстове значення, тут ім'я агентства новин, що опублікувало новину;
- head - текстове значення, заголовок новини, зчитується з RSS;
- link - текстове значення, посилання на повний текст новини, зчитується з RSS;
- text - текстове значення, короткий опис новини, зчитується з RSS;
- pubdate - текстове значення, час публікації новини, зчитується із RSS у спеціальному форматі;
- garbage - зарезервоване поле.

Дані таблиць для Hive можуть зберігатися у різних форматах, одне із них - CSV. Так як MySQL дозволяє імпортувати таблиці в цьому форматі з додатковими налаштуваннями, обираємо саме цей формат.

Також можна застосувати додаткові налаштування імпорту таблиці в MySQL, зокрема - роздільник полів - \t (TAB), роздільник рядків - \n (Enter або перехід рядка), символ екранування - порожнє значення.

Слід зазначити, що символи \t і \n видаляються перед записом до таблиці.

3.2 Створення таблиць Hive

Для роботи з таблицями попередньо необхідно запустити Hadoop, а також командний рядок Hive, аналогічно тому, як це було здійснено у розділі 2. Результат виконання означених дій наводиться на рис.3.2.

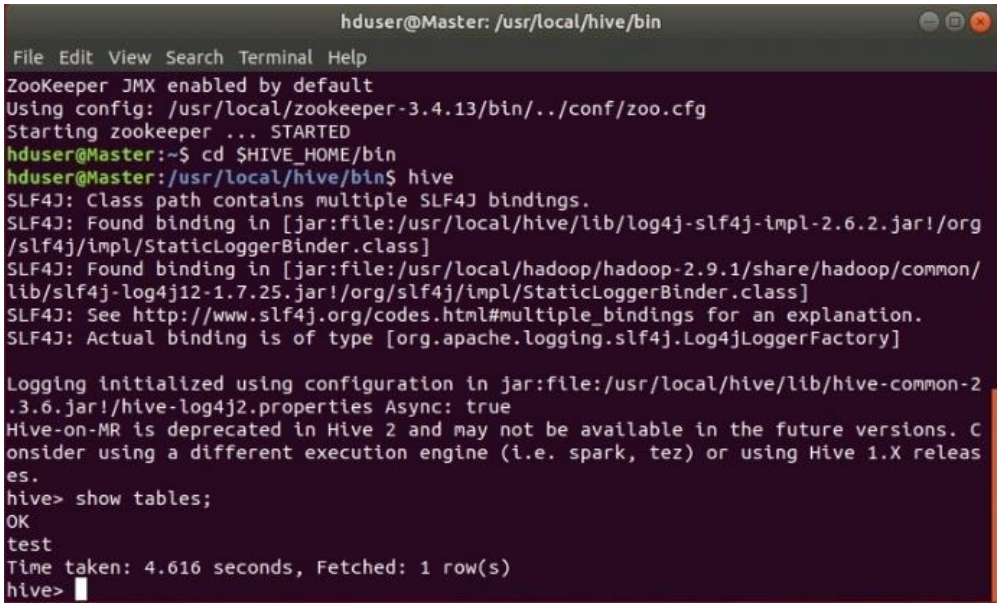


Рисунок 3.2 – Консоль підготовленого робочого середовища

Тепер створимо таблицю Hive, для чого скористаємося наступною командою:

```
create table mynews (ID int, agency String, head String, link String, text String, pubdate String, garbage int) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' LINES TERMINATED BY '\n' STORED AS TEXTFILE;
```

Результат виконання даної команди можемо бачити на рисунку 3.3.

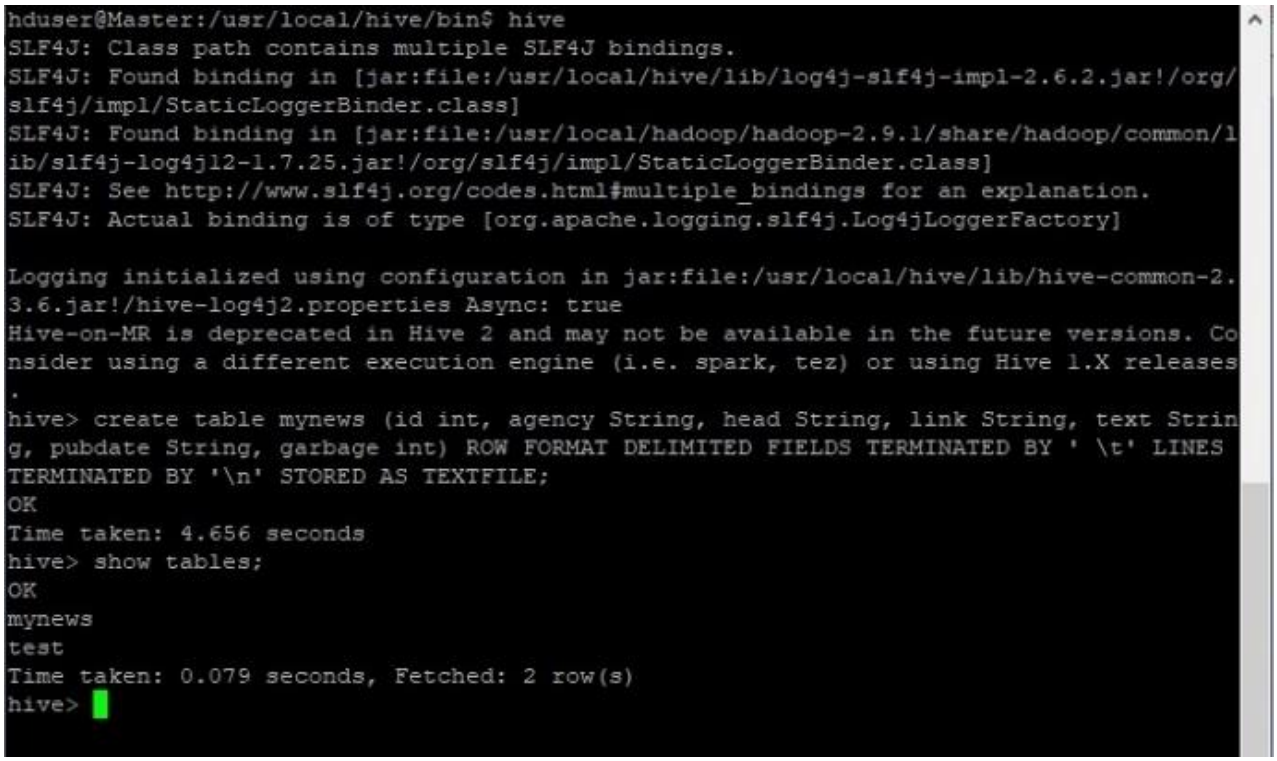


Рисунок 3.3 – Сповіщення про створення таблиці

Для перевірки створеної таблиці можемо скористатися командою [8, 13]:

```
show tables;
```

Слід зазначити, що у HDFS буде створено однойменний каталог, а саме - /user/hive/warehouse/mynews. Перевірку даного факту можна виконати наступною командою:

```
hadoop fs -ls /user/hive/warehouse/
```

Окрім цього, можна також скористатися веб-інтерфейсом, для чого в адресному рядку браузера необхідно вказати IP_адресу майстрер-вузла (або ім'я) та порт. У нашому випадку маємо: master:50070.

Далі у меню Utilities обирається Browse file system [13], як показано на рисунку 3.4.

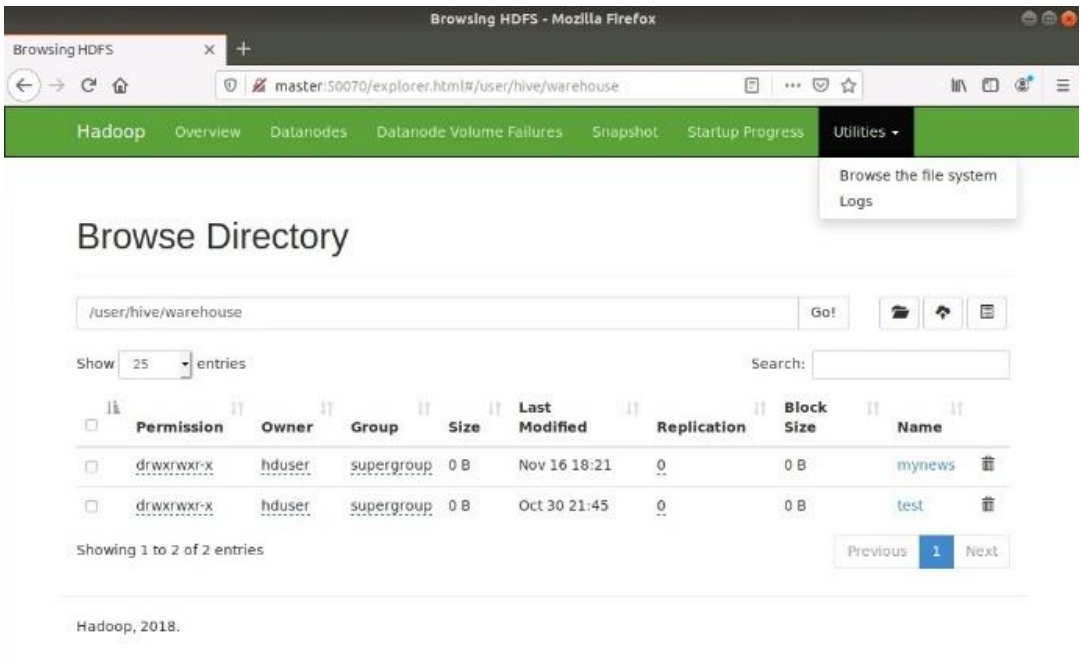


Рисунок 3.4 – Перегляд змісту директорії у браузері

3.3 Виконання SQL-запитів у Hive

Виконаємо запис даних у створену таблицю.

Спершу перевіримо що знаходиться у створеній таблиці за допомогою запиту:

```
select * from mynews;
```

У результаті побачимо, що існує єдиний запис, як показано на рисунку 3.5.

Рисунок 3.5 – Перегляд змісту створеної таблиці

Далі виконаємо перегляд змісту директорії таблиці у HDFS (Рисунок 3.6).

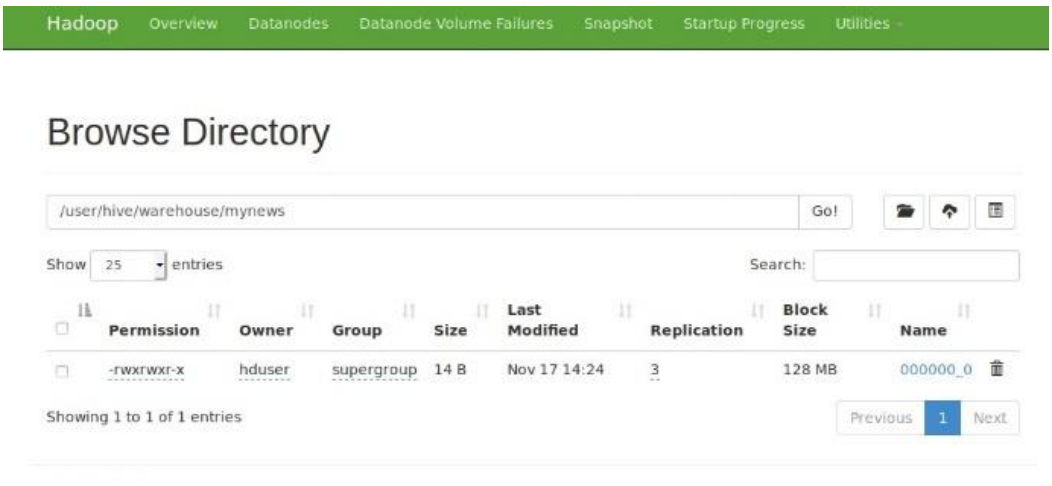


Рисунок 3.6 – Перегляд змісту директорії таблиці у HDFS

Після цього можна завантажити дані з дампу сторонньої таблиці MySQL у таблицю Hive.

При цьому, завантажити дані можна як простим копіюванням csv-файлу HDFS у директорію створеної таблиці, так і через запит Hive, для чого слугує наступна команда:

```
load data local inpath '/tmp/mynes1.csv' into table mynews;
```

У результаті виконання означеної команди дані буде завантажено, файл у сутності було скопійовано до директорії /user/hive/warehouse/mynews/.

Далі в інтерфейсі огляду файлової системи можемо переглянути сам файл і його властивості (рис. 3.7).

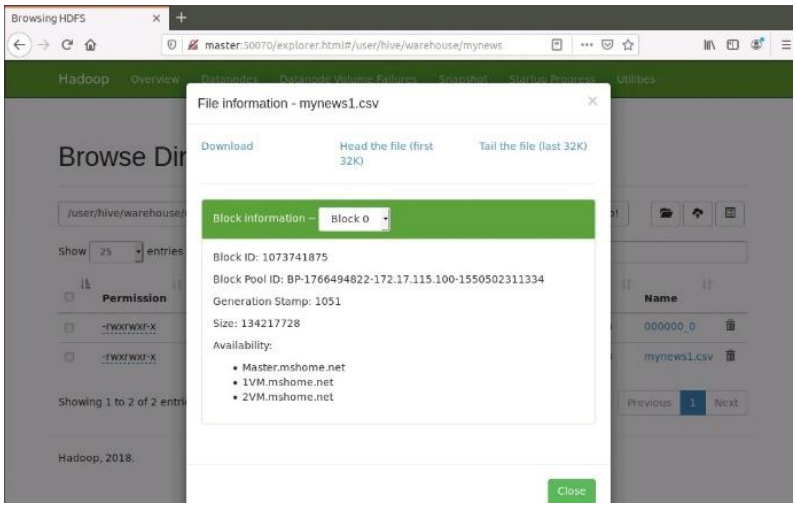


Рисунок 3.7 – Перегляд завантаженого файлу та його властивостей

Далі виконаємо SQL-запит до сформованого csv-файлу (рис.3.8):

```
select * from tables where id=10000;
```

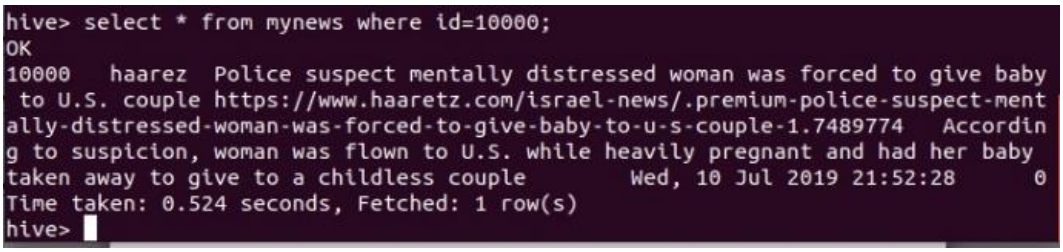


Рисунок 3.8 – Виконання SQL-запиту до сформованого csv-файлу

Далі виконаємо вибірку записів з csv-файлу новин. Для цього створюємо запит з умовою like [13], а саме:

```
select * from mynews where text like '%Trump%';
```

Цей запит читається так: вибрати всі записи з таблиці mynews, де у стовпці text зустрічається комбінація символів Trump.

Новин таких декілька, вивід даних представлено на скріншоті рис.3.9.

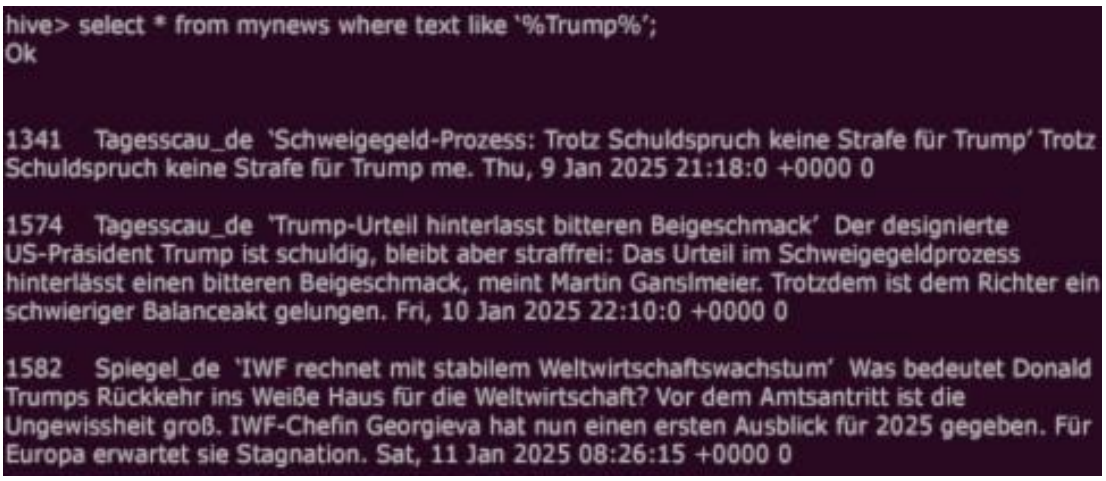


Рисунок 3.9 – Результат виконання запиту з умовою

4 ОПТИМІЗАЦІЯ ФУНКЦІОНУВАННЯ APACHE HIVE

У загальному випадку, якщо не йде мова про виконання в обчислювальному кластері специфічних задач, перелік заходів з оптимізації Apache Hive включає у себе [3, 6, 14]:

− партиціонування даних;
− налаштування пам'яті на етапи MapReduce;
− обробка асиметрії даних в Apache Hive;
− з'єднання таблиць;
− налаштування тайм-ауту завдання.

Розглянемо кожен з цих заходів детально.

4.1 Партіціонування даних

Поділ, або партиціонування даних для Hive являє собою спосіб поділу таблиці на різні частини на основі значення стовпця розділу, спрямований на прискорення виконання запитів до зрізу даних [3, 6, 14].

Партиціонування рекомендується виконувати з самого початку, перед безпосередньо виконання обчислень. Наприклад, якщо виконується аналіз даних про споживчу активність, краще об'єднати дані за один день, після чого додати розділ для цього дня у таблицю Hive. Для цього можна використовувати оператор alter table, щоб вставити нові дані до Hive-таблиці cons_act у вигляді розділу, наприклад [14]:

```
alter table cons_act add partition(date='20240709') location '/activity/consumer/20240709/';
```

Таким чином, виконуючи цей крок щодня, щоб завантажити дані за день у Hive, можна суттєво підвищити продуктивність обчислень.

Окрім того, маючи партиціоновані дані, можна вказати діапазони дат за умови WHERE у SQL-запиті, щоб Hive завантажував лише конкретні дати замість всього діапазону даних. Це зменшить розмір сканованих даних та час виконання запиту. Приклади задання діапазонів дат показано далі [14]:

```
select * from cons_act where date='20240709';
```

```
select * from const_act where date >='20240707' and date
<='20240709';
select * from cons_act where date like '202407%';
select * from cons_act where date like '2024%';
```

4.2 Налаштування пам'яті на етапи MapReduce

Завдання Hive виконуються за обчислювальною моделлю MapReduce, тобто, як набір завдань зіставлення та згортки.

При цьому, за замовчуванням кожне завдання одержує фіксовану пам'ять на кожний етап [3, 6, 14].

Тобто, у зазначених умовах, запустивши на виконання завдання Hive над великими даними, нерідко можна зіткнутися з тим, що проміжні результати обчислень знищуються на етапі зіставлення чи згортки з помилкою нестачі пам'яті. Відтак щоб уникнути подібних ситуацій, необхідно виконати зміну значення конфігурації пам'яті для кожного етапу.

Наприклад, наступні налаштування встановлять пам'ять етапу Map для кожного завдання рівною 8 ГБ а для для Reduce - 10 ГБ [14]:

```
set mapreduce.map.memory.mb=8000;
set mapreduce.reduce.memory.mb=10000;
```

Загальна пам'ять Hive обмежена максимальним обсягом пам'яті контейнера, який, у свою чергу, налаштовується адміністратором кластеру Hadoop.

Разом з тим, якщо встановити значення пам'яті вище максимального обсягу пам'яті контейнера, завдання Hive не запуститься, оскільки це заборонено механізмом управління ресурсами Hadoop.

4.3 Обробка асиметрії даних в Apache Hive

Суттєвим чином на продуктивність обробки даних здійснює вплив перекіс або нерівномірність даних – випадки, коли окремі стовпці або ключі даних мають більше значень, ніж інші дані [14].

У свою чергу, подібна асиметрія даних досить часто зустрічається у наборах великих даних.

Наприклад, у даних з онлайн-ігри, певна вікова група користувачів буде активнішою на платформі. При цьому, навіть у рамках цієї групи може бути виділено окремі підгрупи, активніші в окремі періоди часу.

У ході обробки таких даних з нерівномірним розподілом запити або виконуватимуться повільніше, або матимуть місце збої на етапах згортки, де відбувається агрегування. Наприклад, під час агрегації даних активності користувачів протягом дня, шляхом з'єднання таблиці погодинної активності з таблицею користувачів. У цьому випадку рекомендується налаштувати конфігурацію hive.optimize.skewjoin, щоб Hive сам обробляв таку асиметрію в з'єднанні [14]:

```
set hive.optimize.skewjoin=true;
```

Інший пов'язаний параметр дозволить визначити кількість рядків асиметричного ключа, наприклад, якщо ключ у SQL-запиті (година дня в наведеному вище прикладі) має рядки більше 60000, то Hive розглядатиме його як перекіс у цій операції з'єднання та оброблятиме так, щоб зменшити затримку завдання :

```
set hive.skewjoin.key=60000;
```

4.4 З'єднання таблиць з MapSideJoin

З'єднання таблиць вважається найпоширенішою операцією під час виконання SQL-запитів [6, 14].

Наприклад, дані містяться в декількох таблицях, які потрібно з'єднати по ключу, щоб вибрати потрібні стовпці з них. Якщо дані в таблицях нерівномірні за розміром, може бути використано оператор MapJoin. Зазначена операція застосовується для з'єднання таблиць, одна з яких є невеликою, а інша - навпаки, дуже об'ємною. Розмір маленької таблиці встановлюється через властивість hive.Mapjoin.smalltable.filsize. При цьому, якщо файл таблиці має вагу меншу значення у МБ, встановленого властивістю hive.Mapjoin.smalltable.filsize, то таблиця вважається маленькою. Інакше – навпаки. За замовчуванням значення властивості hive.Mapjoin.smalltable.filsize дорівнює 25 МБ.

Наприклад, вказати меншу таблицю для з'єднання MapJoin можна так [14]:

```
set hive.auto.convert.join=true;
select user_details.user_name, user_details.user_age,
hourly_act.num_of_games, hourly_act.active_min
from user_details join hourly_act on user_details.userid =
hourly_act.user_id;
```

У цьому SQL-запиті таблиця user_details вважається невеликою і містить деякі дані про користувача (ідентифікатор, ім'я та вік), а у великій таблиці hourly activity будуть записи про погодинну активність всіх користувачів у будь-який час дня. Встановлення конфігурації hive.auto.convert.join на значення true вказує Hive на перенесення таблиці user_details у пам’ять, що прискорює виконання запиту.

Додатково можна оптимізувати подібний запит, застосувавши бакетування (bucketing) - метод організації таблиць або розділів в блоки даних. Це найефективніше для вибірки даних, а з'єднання 2-х таблиць, які групуються за одним і тим самим стовпцям, може бути реалізовано MapJoin сегментів (бакетів). У цьому випадку Mapper, який працює із сегментом лівої таблиці, для виконання з'єднання повинен завантажити лише відповідні сегменти правої таблиці. Зазвичай це працює найкраще, якщо таблиці розділені по стовпцю, що з'єднується, і відсортовані по цьому стовпцю, тому що тоді сортування злиттям стає ефективним.

4.5 Тайм-аут завдання

Ще один важливий параметр під час роботи з великими даними - тайм-аут на рівні завдань MapReduce [3, 6, 14].

Даний параметр особливо важливий під час виконання SQL-запитів з умовами WHERE та GROUP BY. Запити такого типу у поєднанні із проблемами асиметрії даних іноді призводять до того, що деякі завдання MapReduce виконуються дуже довго. Це призводить до того, що завдання, що виконується, пропускає heartbeat-повідомлення майстру програми, тобто, головному вузлу, який виконує моніторинг стану завдань усього кластеру Hive. Майстер очікує heartbeat-повідомлень від вузлів кластера кожні 10 хвилин (за замовчуванням 600000 мілісекунд). Якщо цього недостатньо, можна встановити більше значення конфігурації mapred.task.timeout, наприклад 3600000 мілісекунд, тобто. 30 хв:

```
set mapred.task.timeout=3600000;
```

За зазначених умов кожне завдання в Hive-завданні може виконуватися протягом 30 хвилин, перш ніж буде відправлено heartbeat-повідомлення майстру програми.

Це гарантує, що завдання не буде знищено через те, що це повідомлення не надходить до майстра програми кожні 10 хвилин, а продовжує очікувати збільшеного періоду часу (тайм-аут), рівного 30 хвилин перед завершенням.

4.6 Оптимізація запитів у Hive

Перед здійсненням кожного запиту, вбудований оптимізатор Hive (Cost-Based Optimizer, CBO) створює план виконання, що являє собою послідовність завдань, які мають бути виконані, щоб отримати необхідні дані найбільш ефективним способом [15].

У свою чергу, для перегляду планів виконання, а також отримання детальної статистики щодо роботу Hive для подальшої оптимізації запитів використовуються внутрішні команди Hive - explain та analyze.

Для перегляду прикладів оптимізації запитів створимо тестові Hive-таблиці transactions та accounts. При цьому, для створення та заповнення тестових таблиць виконаємо наступний SQL за допомогою /bin/beeline [15]:

```
CREATE DATABASE IF NOT EXISTS explain_demo;
USE explain_demo;

DROP TABLE IF EXISTS transactions;
CREATE TABLE transactions(txn_id int, acc_id int, txn_amount
decimal(10,2)) PARTITIONED BY (txn_date date);
INSERT INTO transactions VALUES
(1, 1002, 10.00, '2025-01-01'),
(2, 1002, 20.00, '2025-01-03'),
(3, 1002, 30.00, '2025-01-02'),
(4, 1001, 100.50, '2025-01-02'),
(5, 1001, 150.50, '2025-01-04'),
(6, 1001, 200.50, '2025-01-03'),
(7, 1003, 50.00, '2025-01-03'),
(8, 1003, 50.00, '2025-01-01'),
(9, 1003, 75.00, '2025-01-04');
```

```
DROP TABLE IF EXISTS accounts;
CREATE TABLE accounts(id int, full_name string);
INSERT INTO accounts VALUES
(1001, 'John Smith'),
(1002, 'Sarah Connor'),
(1003, 'Rick Sanchez');
```

Отримаємо таблиці, структуру яких описано далі:

```
SELECT * FROM transactions;
+---------------------+---------------------+-------------------------+-----------------------+
| transactions.txn_id | transactions.acc_id | transactions.txn_amount | transactions.txn_date |
+---------------------+---------------------+-------------------------+-----------------------+
| 1                   | 1002                | 10.00                   | 2025-01-01            |
| 8                   | 1003                | 50.00                   | 2025-01-01            |
| 3                   | 1002                | 30.00                   | 2025-01-02            |
| 4                   | 1001                | 100.50                  | 2025-01-02            |
| 2                   | 1002                | 20.00                   | 2025-01-03            |
| 6                   | 1001                | 200.50                  | 2025-01-03            |
| 7                   | 1003                | 50.00                   | 2025-01-03            |
| 5                   | 1001                | 150.50                  | 2025-01-04            |
| 9                   | 1003                | 75.00                   | 2025-01-04            |
+---------------------+---------------------+-------------------------+-----------------------+

SELECT * FROM accounts;
+-------------+--------------------+
| accounts.id | accounts.full_name |
+-------------+--------------------+
| 1001 | John Smith |
| 1002 | Sarah Connor |
| 1003 | Rick Sanchez |
+-------------+--------------------+
```

У випадку використання команди `explain` отримаємо план виконання (execution plan) Hive-запиту. Вивід даних при цьому включає докладний опис того, якими етапами Hive планує виконати запит. Комадна має наступний синтаксис:

```
EXPLAIN
[EXTENDED|DEPENDENCY|AUTHORIZATION|VECTORIZATION|FORMATTE
D] <query>
```

Розлянемо окремо значення ключів команди:

– EXTENDED - Повертає розширену інформацію про план виконання.

– DEPENDENCY — містить докладну інформацію про таблиці та партії, які задіяні у запиті.

– AUTHORIZATION — повертає інформацію про всі об'єкти, які потребують авторизації, включаючи проблеми з правами доступу (якщо є).

– VECTORIZATION — повертає інформацію про ті Map/Reduce-завдання, для яких не застосовувалася векторизація.

FORMATTED - Форматує висновок в JSON-форматі.

<query> - Запит, який необхідно аналізувати.

Виконаємо команду, поєднавши її з командою створення таблиці звіту:

```
CREATE TABLE IF NOT EXISTS top_txns_per_acc(acc_id int,
max_txn_value decimal(10,2));

EXPLAIN
    FROM transactions INSERT OVERWRITE TABLE top_txns_per_acc
    SELECT acc_id, MAX(txn_amount) GROUP BY acc_id;
```

Отримаємо наступний результат [15]:

```
+---------------------------------------------- ---+
| Explain |
+---------------------------------------------- ---+
| Plan optimized by CBO. |
| |
| Vertex dependency in root stage |
| Reducer 2 <- Map 1 (SIMPLE_EDGE) |
| Reducer 3 <- Reducer 2 (CUSTOM_SIMPLE_EDGE) |
| |
| Stage-3 |
| Stats Work{} |
|Stage-0 |
| Move Operator |
| table:{"name:":"explain_demo.top_txns_per_acc"} |
| Stage-2 |
| Dependency Collection{} |
| Stage-1 |
| Reducer 3 |
|File File Output Operator [FS_13] |
| Group By Operator [GBY_11] (rows=1 width=552) |
|Output:["_col0","_col1"],aggregations:["compute_stats(VALUE._col0)","comp
ute_stats(VALUE._col1)"] |
|<-Reducer 2 [CUSTOM_SIMPLE_EDGE] |
| File Output Operator [FS_6] |
|table:{"name:":"explain_demo.top_txns_per_acc"} |
| Group By Operator [GBY_4] (rows=4 width=12) |
|
Output:["_col0","_col1"],aggregations:["max(VALUE._col0)"],keys:KEY._col0 |
| <-Map 1 [SIMPLE_EDGE] vectorized |
```

```
      | SHUFFLE [RS_16] |
      | PartitionCols:_col0 |
      |Group By Operator [GBY_15] (rows=9 width=12) |
      | Output:["_col0","_col1"],aggregations:["max(txn_amount)"],keys:acc_id |
      | Select Operator [SEL_14] (rows=9 width=12) |
      | Output:["acc_id","txn_amount"] |
      | TableScan [TS_0] (rows=9 width=12) |
      |
explain_demo@transactions,transactions,Tbl:COMPLETE,Col:NONE,Output:["acc_id","t
xn_amount"] |
      | PARTITION_ONLY_SHUFFLE [RS_10] |
      | Group By Operator [GBY_9] (rows=1 width=536) |
      |            Output:["_col0","_col1"],aggregations:["compute_stats(acc_id,
'hll')","compute_stats(max_txn_value, 'hll')"] |
      | Select Operator [SEL_8] (rows=4 width=12) |
      | Output:["acc_id","max_txn_value"] |
      | Refer to previous Group By Operator [GBY_4] |
      | |
      +-------------------------------------------------- ---+
```

Результат виводу даних `explain` може значно відрізнятися залежно від запиту, але основними інформаційними блоками є:

– Stage dependencies. Містить перелік етапів (stages), які разом формують загальний план виконання. Окремий етап може бути Map/Reduce-завданням, операцією читання/запису HDFS, зверненням до Hive Metastore і так далі.

Stage plans. Кожен блок описує деталі окремого етапу, включаючи задіяні оператори, порядок сортування тощо.

ВИСНОВКИ

Відповідно до технічного завдання, у ході виконання кваліфікаційної роботи було виконано:

− дослідження архітектури, можливостей та загальних прийомів роботи з засобом Apache Hive;

− огляд процесу розгортання та подальшого конфігурування платформи Apache Hive у межах обчислювального кластеру з підготовкою до безпосередньо використання;

− дослідження закономірностей та специфіки роботи з Big Data, представлених у структурованому табличному вигляді, використовуючи для цього Apache Hive;

− огляд поширених підходів до оптимізації обробки Великих Даних на базі Apache Hive у середовищі Hadoop, включаючи такі заходи, як партиціонування даних, злиття даних, обробка асиметрії даних, налаштування пам'яті на етапи MapReduce, а також налаштування тайм-аутів виконання завдань.

Дослідження виконувалося на прикладі обчислювального кластеру під управлінням Linux Ubuntu 22 LTS.

У ході цього виявлено, що:

− інтеграція Apache Hive у кластер Hadoop не вимагає суттєвих витрат часу на налаштування, оскільки зводиться до встановлення засобу виключно на майстер-вузол кластеру;

− Apache Hive дозволяє виконувати Map/Reduce операції відносно структурованих Великих даних на засадах SQL-доступу, що суттєво скорочує час доступу а відтак – самих обчислень також;

− Apache Hive дозволяє виконати прив’язку обчислювального кластеру до СУБД різних типів, що дозволяє напряму отримувати доступ до банків даних з можливістю швидких обчислень та без необхідності використання будь-яких додаткових модулів конвертації форматів представлення даних;

− робота зі структурованими даними виконується не безпосередньо на рівні їх джерел, а з попереднім розміщенням СУБД у межах файлової системи HDFS, що також сприяє збільшенню швидкості виконання запитів а відтак – усього циклу Map/Reduce обчислень.

Таким чином, усі завдання кваліфікаційної роботи виконано у повному обсязі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Big Data: Introduction to On-Premise and Cloud-Based Solutions [Електронний ресурс] – Режим доступу: https://medium.com/@luvverma2011/big-data-part-1-94bf087cad46.

2. 20 Best Big Data Applications & Examples in Today's World [Електронний ресурс] – Режим доступу: https://www.ubuntupit.com/best-big-data-applications-in-todays-world/.

3. Apache Hive [Електронний ресурс] – Режим доступу: https://hive.apache.org/.

4. Explained: Apache Hive. An overview of Hive [Електронний ресурс] – Режим доступу: https://medium.com/@john_tringham/explained-apache-hive-5c801f543cb6.

5. T. White. Hadoop:The Definitive Guide. - Sebastopol: O'Reilly, 2015 - 765 p. ISBN: 978-1-491-90163-2

6. S. Shaw, A. François, A. Gupta. Practical Hive. A Guide to Hadoop's Data Warehouse System. - Springer Nature, 2016 - 292 p. ISBN: 9781484202722

7. D. Lee. Instant Apache Hive Essentials How-to. - Birmingham: Packt Publishing, 2024. - 565 p.

8. E. Capriolo, D. Wampler, J Rutherglen. Programming Hive: Data Warehouse and Query Language for Hadoop. - O'Reilly, 2012. - 350 p.

9. Apache Hive Tutorial – A Single Best Comprehensive Guide [Електронний ресурс] – Режим доступу: https://data-flair.training/blogs/hive-tutorial/

10. Hive: Simple installation and construction of derby and partial operations (Quick Start) [Електронний ресурс] – Режим доступу: https://programmersought.com/article/63396245346/

11. MySQL on Linux Tutorial [Електронний ресурс] – Режим доступу: http://www.yolinux.com/TUTORIALS/LinuxTutorialMySQL.html

12. Hive Thrift Service [Електронний ресурс] – Режим доступу: https://www.oreilly.com/library/view/programming-hive/9781449326944/ch16.html

13. HiveSQL explained. Unlocking Data Insights: Understanding HiveSQL's Role in AI, ML, and Data Science [Електронний ресурс] – Режим доступу: https://aijobs.net/insights/hivesql-explained/

14. 7 Best Hive Optimization Techniques – Hive Performance [Електронний ресурс] – Режим доступу: https://data-flair.training/blogs/hive-optimization-techniques/

15. 15 Hive Query to Unlock the Power of Big Data Analytics [Електронний ресурс] – Режим доступу: https://www.analyticsvidhya.com/blog/2020/12/basic-and-highly-used-hive-queries-that-all-data-engineers-must-know/