BIG DATA ANALYSIS ON HEALTHCARE INSURANCE COMPANY

CS6003 – BIG DATA ANALYTICS

Submitted by

I Sheeba Grace (2020103047)

Swathi M (2020103053)

in partial fulfillment of the requirements for the award of thedegree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

COLLEGE OF ENGINEERING, GUINDY

ANNA UNIVERSITY: CHENNAI 600 025

MAY 2023

ANNA UIVERSITY: CHENNAI 600025

BONAFIDE CERTIFICATE

Certificate that this project request titled Analysis on Healthcare Insurance Company is the bonafide work of **I Sheeba Grace(2020103047), Swathi M(2020103053)** who carried out the project work under my supervision, for the fulfillment of the requirements as part of the CS6003 – Big Data Analytics

Dr. S. Valli

HEAD OF THE DEPARTMENT

Professor

Department of Computer Science and Engineering

College of Engineering Guindy
12, Sardar Patel Rd, Guindy
Chennai, Tamil Nadu - 600025

Dr. R. Arockia Xavier Annie

SUPERVISOR

Assistant Professor

Department of Computer Science and Engineering

College of Engineering Guindy
12, Sardar Patel Rd, Guindy
Chennai, Tamil Nadu - 600025

ABSTRACT

Big data analytics in insurance companies involve the use of advanced technology and tools to analyze vast amounts of data generated within the insurance industry. The application of big data analytics in the insurance industry can help insurers to better understand their customers, provide more personalized insurance products, and improve the overall customer experience. The analysis of big data in insurance can also help to identify fraudulent claims, manage risk more effectively, and improve underwriting decisions. Additionally, the use of big data analytics can help insurers to optimize their operations and drive their growth. In our project,we carry out the analysis using various big data tools like Hadoop,Spark,Sqoop etc.Initially the preprocessed dataset is stored in Relational Database Management System(RDBMS).To do analysis the data in RDBMS is transferred to Hadoop using sqoop. After uploading the data in to HDFS we connect to spark. Here we analyze the data with help of python. Finally we are expected to get our desired result in tabular form and that result is used to visualize our use cases.

ACKNOWLEDGEMENT

Foremost, we would like to express our sincere gratitude to our project guide, **Dr.R.Arockia Xavier Annie**, Assistant Professor, Department of Computer Science and Engineering, College of Engineering Guindy, Chennai for her constant source of inspiration. We thank her for the continuous support and guidance which was instrumental in taking the project to successful completion.

We are grateful to **Dr. S. Valli** , Professor and Head, Department of Computer Science and Engineering, College of Engineering Guindy, Chennai for her support and for providing necessary facilities to carry out for the project.

We would also like to thank our friends and family for their encouragement and continued support. We would also like to thank the Almighty for giving us the moral strength to accomplish our task.

I SHEEBA GRACE **SWATHI M**

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

A healthcare insurance company is seeking to boost its revenue and gain a better understanding of its customers. To achieve this, it plans to leverage the power of the Big Data Ecosystem to analyze competitors' data collected from various sources such as web scraping and third-party sources. The aim of this analysis is to track customer behavior and conditions, enabling the company to tailor insurance policy offers to individual needs and calculate royalties for prior policyholders. By doing so, the company hopes to increase its revenue and gain a better understanding of its customers to enhance its services.

1.1 OBJECTIVES

The goal of the project is to create data pipelines for the Health Care insurance company which will make the company make appropriate business strategies to enhance their revenue by analyzing customers behaviors and send offers and royalties to customers respectively.

1.2 PROBLEM STATEMENT

Healthcare companies face challenges in leveraging big data analysis to improve revenue and gain deeper insights into customer behavior. These organizations aim to harness the power of the Big Data Ecosystem to analyze vast amounts of data collected from various sources, including web scraping and third-party data. However, the current problem lies in the lack of efficient methods to extract meaningful information from this extensive data pool and utilize it effectively. The healthcare industry requires an advanced analytics solution that can process and interpret the collected data to track customer behavior and conditions accurately. By doing so, insurance companies can tailor their insurance policy offers to individual needs and calculate royalties for prior policyholders more accurately. This data-driven approach is expected to enhance revenue generation and improve customer understanding, ultimately leading to better service provision.

1.3 NEED FOR THE SYSTEM

By leveraging the power of the Big Data Ecosystem to analyze competitors' data and track customer behavior, insurance companies can identify patterns and trends that can be used to tailor insurance policy offers to individual needs. This personalized approach can lead to higher customer acquisition, retention, and increased revenue.

Utilizing big data analytics can provide valuable insights into customer behavior and conditions, allowing insurance companies to optimize their services. By understanding customers' needs and expectations better, companies can enhance the overall customer experience and provide proactive support, leading to higher customer satisfaction and loyalty.

Calculating royalties for prior policyholders accurately is a crucial aspect of the insurance business. By leveraging big data analytics, companies can analyze historical data and customer behavior to determine the appropriate royalties, reducing errors and ensuring fair compensation for policyholders.

1.4 CHALLENGES IN THE SYSTEM

Implementing a big data analytics system in healthcare insurance companies is met with challenges such as managing large data volumes, ensuring data security, integrating diverse data sources, analyzing complex datasets, scaling infrastructure, addressing talent shortage, and managing ethical considerations and biases. Overcoming these challenges is crucial to leverage the potential of big data for improved decision-making, personalized offerings, and enhanced customer satisfaction, ultimately leading to increased revenue and a better understanding of customers in the dynamic healthcare insurance industry.

1.5 SCOPE OF THE PROJECT

The scope of the project involves implementing a big data analytics system in a healthcare insurance company. The project encompasses collecting and managing large volumes of data from various sources, such as web scraping and third-party data. It includes developing a secure and scalable infrastructure to store and process the data. The project also involves designing and implementing advanced analytics techniques to analyze the data and extract valuable insights related to customer behavior and conditions.

CHAPTER 2

LITERATURE SURVEY

2.1Big Data Analytics

The paper [1] proposes the idea of implementing big data technologies in healthcare system. . The volume of data available in digital form pertaining to the health of individuals has already become huge and is rapidly growing. It includes both the traditional clinically collected data, administrative data and increasingly other health and contextual sensor data collected in non-clinical settings. While the importance of analytics to healthcare is increasingly recognized, the healthcare industry is yet to fully harness the emergence of big data technologies specifically for carrying out analytics and to fully recognize the implications for healthcare.In this paper they introduced a framework to help organize and guide the understanding of the application of big data technologies for processing health and healthcare data. The framework includes four layers: Health Data Sources, the Big Data Technology Layer, the Big Data Analytics Layer, and the Applications Layer. Once data is collected and uploaded into the big data technology platform, analytics, data mining or machine learning approaches and technologies can be applied.

2.2 Big Data Applications

The paper[2] proposes the application of big data technologies in insurance companies. Big data in a narrow sense refers to the extraction of valuable, massive, and diversified transaction data, interactive data, sensor data, etc. through its rapid acquisition, processing, and analysis. The data involved are so large that they cannot be collected, processed, or managed in a reasonable time frame by existing technology and tools.Therefore, they require a new processing mode,decision-making ability, insight and discovery ability, and process optimization ability. In traditional insurance product innovation it is easy toignore the information derived from the behavior of individual consumers, so it is difficult to launch personalized products. Moreover, the data held by insurance companies are mainly internal data, but rarely contain external customer behavior data. Through big data, insurance companies can analyze user needs, reduce product development costs, solvet he problem of product development cycle lengths being too long to keep up with consumer needs, and develop products more scientifically according to consumer behavior data.This can be done to achieve

more interaction with customers,to provide products and services closer to customer needs,and to develop innovative, personalized products for consumers.

2.3 Summary:

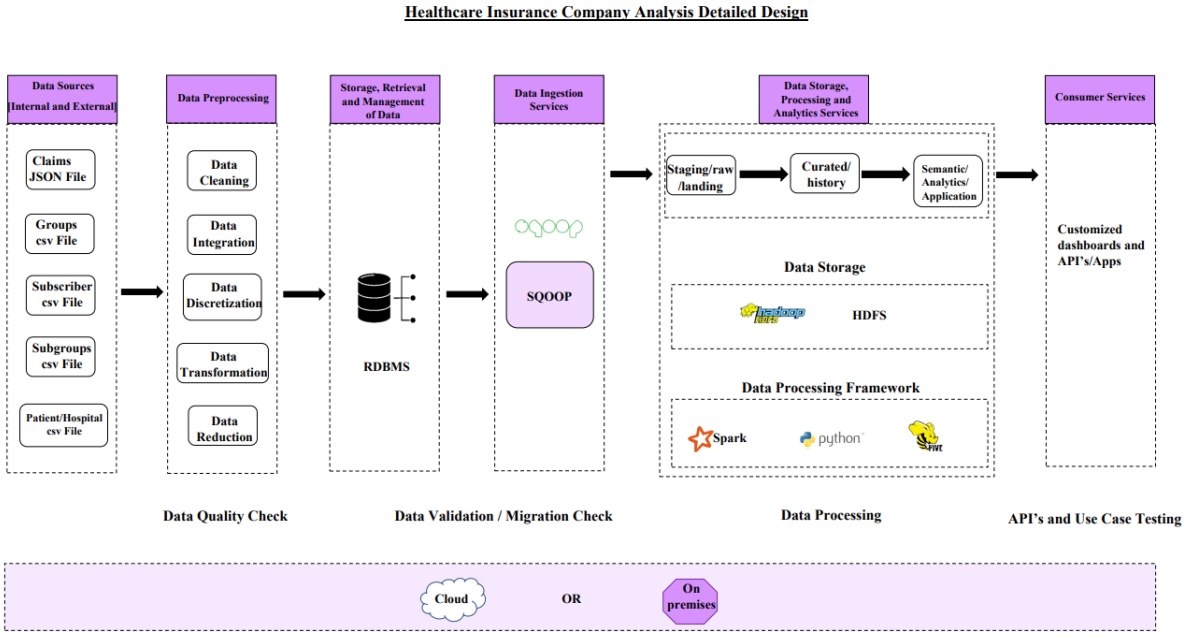
Overall in this research, big data analytics system in a healthcare insurance company offered tremendous potential for revenue growth and enhanced customer understanding. By addressing challenges such as data volume management, security, integration, analysis complexity, scalability, talent shortage, and ethical considerations, the project can pave the way for personalized insurance policy offerings, accurate royalty calculations, and improved service provision. Leveraging the power of big data analytics, the system enables the company to track customer behavior and conditions, leading to tailored insurance solutions and increased revenue. Additionally, the project promotes better decision-making through advanced analytics techniques, while adhering to data privacy regulations and ethical standards. Ultimately, the successful implementation of the project contributes to a better understanding of customers and an improved overall customer experience in the healthcare insurance industry.

CHAPTER 3

SYSTEM DESIGN

3.1 SYSTEM ARCHITECTURE

The system is divided into 6 modules namely data sources,data preprocessing,storage,retrieval and management of data,data ingestion services,data storage and analytics services and consumer services.



3.2 SYSTEM REQUIREMENTS

The system requirements for implementing a big data analytics system in a healthcare insurance company include robust data storage and processing capabilities, support for data integration and ETL processes, strong security and privacy measures, advanced analytics and machine learning capabilities, scalability and performance optimization, user-friendly interfaces for data exploration and visualization, integration with existing systems, backup and disaster recovery mechanisms, compliance with regulatory requirements and data governance practices, and provision of training and ongoing support. These requirements are essential for building a comprehensive and efficient system that can handle large volumes of data, derive meaningful insights, ensure data security and privacy, and meet industry-specific needs.

3.2.1 FUNCTIONAL REQUIREMENTS:

3.2.1.1 Hardware Requirements

- Processor: The hardware should consist of a processor with sufficient processing power to handle the bigdata tools involved in the project. A multi-core processor such as Intel Core i3 or i5 or i7 would be suitable.
- Memory: The hardware should have sufficient memory to handle the large data involved in the project. A minimum of 8GB of RAM is recommended.
- Graphics Card: The hardware should have a graphics card with dedicated GPU memory to accelerate the installation of Hadoop and other tools. A graphics card with a minimum of 2GB VRAM is recommended.
- Storage: The hardware should have sufficient storage capacity to store the big data. A minimum of 256GB of storage is recommended.
- Connectivity: The hardware should have connectivity options such as Wi-Fi, Bluetooth, or Ethernet to transfer data from external sources.
- Dataset: It includes 6 csv files and 1 json file related to healthcare insurance company.

3.2.1.2 Software Requirements

- Programming language: The project should be implemented using a programming language suitable big data analytics such as Python.
- Development environment: The project should be developed using a suitable development environment such as Hadoop 3.3.5,spark1.4.4,hive3.1.2 and sqoop 3.4.0
- Documentation: The project should include detailed documentation on how to install, configure, and run the project, as well as the implementation details of the various big data technologies like Hadoop,spark sql,sqoop and hive.
- Testing framework: The project should use a testing framework such as pytest to automate testing and ensure code quality.
- Deployment: The project should be deployable in a cloud-based or local environment.

3.2.2 NON-FUNCTIONAL REQUIREMENTS:

The following are some potential non-functional requirements in terms of performance for the project “Analysis on Healthcare Insurance Company”.

3.2.2.1 Performance: The project should be optimized for efficient processing and response time. The pre-processing of the dataset should be implemented in a manner such that it reduces the computation time and the response time and does not slow down the analysis of the data. The project should be able to handle data from various sources and provide a reasonably correct output with valuable insights.

CHAPTER 4

MODULE DESCRIPTION

4.1 DATA PRE-PROCESSING:

• After parsing and inferring the schema of the given xml and csv formats, data is ingested.

• We did general data cleaning steps like empty string replacements with actual NULL, data type checks (including date format) and corrections/ rejections, file name checks, empty file checks, malformed record checks and rejection etc.

4.2 STORAGE, RETRIEVAL AND MANAGEMENT OF DATA:

• After the files data is preprocessed, the data model for RDBMS is created so that the files data can be stored into the RDBMS.

• After storing the data into the RDBMS, we now transform according to our business requirements.

Schema Design for SQL Database:

4.3 DATA INGESTION SERVICES:

• Data is then moved to HDFS using the data ingestion tool, Sqoop.

• Sqoop is used for the efficient transfer of data from RDBMS to HDFS and vice versa.

4.4 DATA STORAGE, PROCESSING AND ANALYTICS SERVICES:

• These are carried out using technologies such as HDFS for data storage and Spark, Python and Hive for data processing.

• Data landed to HDFS needs to be analyzed by some analytical queries.

CHAPTER 5

RESULTS & IMPLEMENTATION

5.1 DATA SOURCES:

• Data coming from third-party sources reside in local directory and has csv and

json format.

• Fields present in the data files and tables.

• Data files contain the below fields:

Column Name/Field Name Column Description/Field Description:

Json File Fields

● CLAIM_ID

● PATIENT_ID

● DISEASE_NAME

● SUB_ID

● CLAIM_DATE

● CLAIM_TYPE

● CLAIM_AMOUNT

● CLAIMED_OR_REJECTED

CSV File 1 Fields (Patient.csv)

● PATIENT_ID

● PATIENT_NAME

● PATIENT _GENDER

● PATIENT _BIRTHDATE

● PATIENT _PHONE

● HOSPITAL_ID

● DISEASE _ NAME

● CITY

CSV File 2 Fields (Subscriber.csv)

● SUB_ID

● FIRST_NAME

● LAST_NAME

● STREET

● BIRTH_DATE

● GENDER

● PHONE_NO

● COUNTRY

● CITY

● ZIP_CODE

● SUBGRP_ID

● ELIG_IND

● E_DATE

● T_DATE

CSV File 3 Fields (Group.csv)

- GRP_ID
- GRP_NAME
- PREMIUM_WRITTEN
- GRP_TYPE
- PIN_CODE
- CITY
- COUNTRY
- ESTABLISHMENT_YEAR

CSV File 4 Fields (disease.csv)

- SUBGRP_ID
- DISEASE_NAME
- DISEASE_ID

CSV File 5 Fields (subgroup.csv)

- SUBGRP_ID
- SUBGRP_NAME
- GRP_ID

CSV File 6 Fields (hospital.csv)

- HOSPITAL_ID

● HOSPITAL _ NAME

● CITY

● STATE

● COUNTRY

CSV File 7 Fields (grpsubgrp.csv)

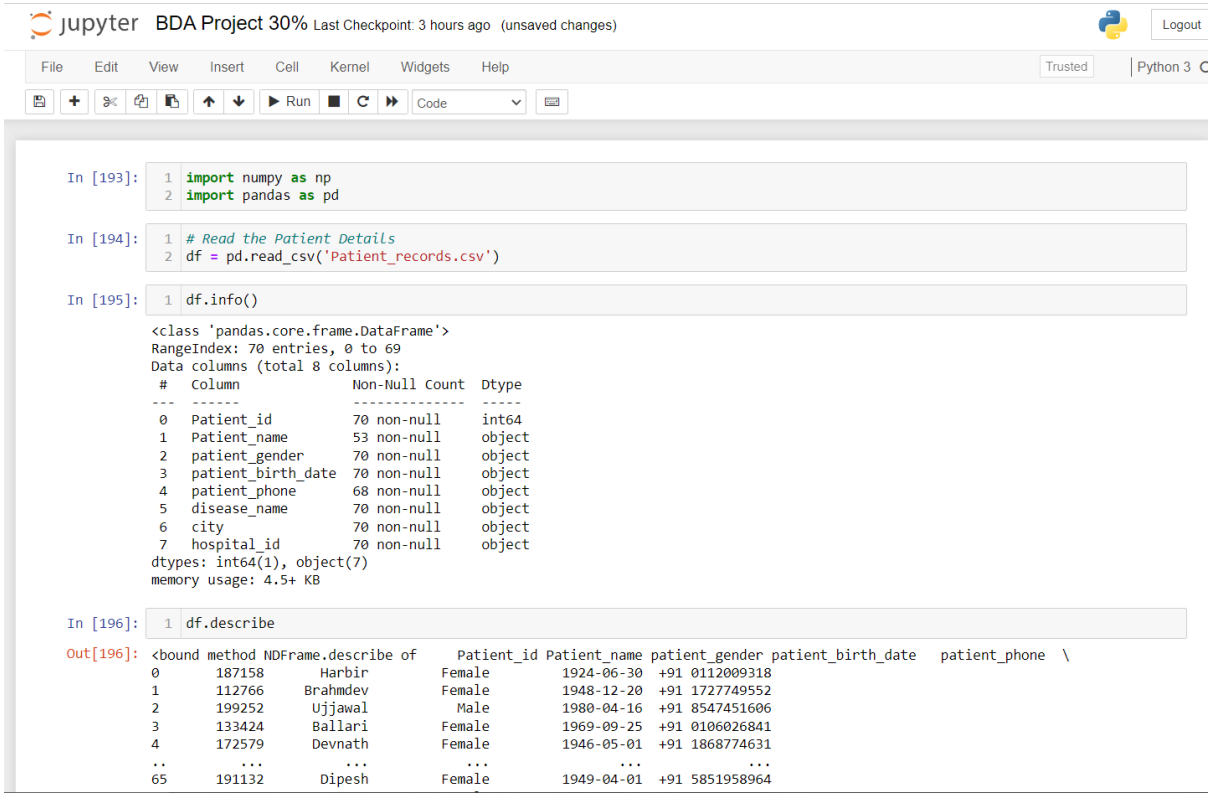
● SUBGRP_ID

● GRP_ID

5.2. DATA PRE-PROCESSING:

The JSON and csv files are preprocessed to make them suitable for further application.

The dataset is loaded and its structure is viewed.



```
68        Diabetes     Ambarnath       H1014
69     Pet allergy       Sonipat       H1017

[70 rows x 8 columns]>
```

```
In [197]:  1 df.head(10)
```

Out[197]:

	Patient_id	Patient_name	patient_gender	patient_birth_date	patient_phone	disease_name	city	hospital_id
0	187158	Harbir	Female	1924-06-30	+91 0112009318	Galactosemia	Rourkela	H1001
1	112766	Brahmdev	Female	1948-12-20	+91 1727749552	Bladder cancer	Tiruvottiyur	H1016
2	199252	Ujjawal	Male	1980-04-16	+91 8547451606	Kidney cancer	Berhampur	H1009
3	133424	Ballari	Female	1969-09-25	+91 0106026841	Suicide	Bihar Sharif	H1017
4	172579	Devnath	Female	1946-05-01	+91 1868774631	Food allergy	Bidhannagar	H1019
5	171320	Atasi	Male	1967-10-02	+91 9747336855	Whiplash	Amravati	H1013
6	107794	Manish	Male	1967-06-06	+91 4354294043	Sunbathing	Panvel	H1004
7	130339	Aakar	Female	1925-03-05	+91 2777633911	Drug consumption	Bihar Sharif	H1000
8	110377	Gurudas	Male	1945-05-06	+91 1232859381	Dengue	Kamarhati	H1001
9	149367	NaN	Male	1925-06-12	+91 1780763280	Head banging	Bangalore	H1013

```
In [198]:  1 df.isnull()
```

Out[198]:

	Patient_id	Patient_name	patient_gender	patient_birth_date	patient_phone	disease_name	city	hospital_id
0	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False

The null values in the dataset are counted and replaced with “NA”

```
In [199]:  1 # Count the total Null values
           2 df.isnull().sum()
```

```
Out[199]: Patient_id            0
          Patient_name         17
          patient_gender        0
          patient_birth_date    0
          patient_phone         2
          disease_name          0
          city                  0
          hospital_id           0
          dtype: int64
```

```
In [200]:  1 # Replace the null values
           2 df["Patient_name"].fillna("NA", inplace = True)
```

```
In [201]:  1 df.head()
```

Out[201]:

	Patient_id	Patient_name	patient_gender	patient_birth_date	patient_phone	disease_name	city	hospital_id
0	187158	Harbir	Female	1924-06-30	+91 0112009318	Galactosemia	Rourkela	H1001
1	112766	Brahmdev	Female	1948-12-20	+91 1727749552	Bladder cancer	Tiruvottiyur	H1016
2	199252	Ujjawal	Male	1980-04-16	+91 8547451606	Kidney cancer	Berhampur	H1009
3	133424	Ballari	Female	1969-09-25	+91 0106026841	Suicide	Bihar Sharif	H1017
4	172579	Devnath	Female	1946-05-01	+91 1868774631	Food allergy	Bidhannagar	H1019

```
In [202]:  1 # Replace the null values
           2 df['patient_phone'].fillna("NA", inplace = True)
```

```
In [203]:  1 df.head()
```

Out[203]:

	Patient_id	Patient_name	patient_gender	patient_birth_date	patient_phone	disease_name	city	hospital_id

The duplicate values are checked and dropped, if any.

In [204]:
```
# Count the total Null values
df.isnull().sum()
```

Out[204]:
```
Patient_id            0
Patient_name          0
patient_gender        0
patient_birth_date    0
patient_phone         0
disease_name          0
city                  0
hospital_id           0
dtype: int64
```

In [205]:
```
# Check the duplicates
df.duplicated()
```

Out[205]:
```
0     False
1     False
2     False
3     False
4     False
      ...
65    False
66    False
67    False
68    False
69    False
Length: 70, dtype: bool
```

In [206]:
```
# Drop duplicates
df.drop_duplicates(subset=None, keep='first', inplace=False, ignore_index=False)
```

Out[206]:

	Patient_id	Patient_name	patient_gender	patient_birth_date	patient_phone	disease_name	city	hospital_id
0	187158	Harbir	Female	1924-06-30	+91 0112009318	Galactosemia	Rourkela	H1001
1	112766	Brahmdev	Female	1948-12-20	+91 1727749552	Bladder cancer	Tiruvottiyur	H1016

The patient ID is validated and the preprocessed data is exported. Similar process is applied to all other JSON and csv files.

66	105686	NA	Male	1930-09-01	+91 7061843400	Hepatitis	Kolhapur	H1008
67	160140	Kishan	Male	1923-05-12	+91 9067652693	Rett Syndrome	Srikakulam	H1002
68	114252	NA	Female	1927-02-26	+91 4984346995	Diabetes	Ambarnath	H1014
69	188365	Bhageeratha	Male	1973-03-21	+91 0590662722	Pet allergy	Sonipat	H1017

70 rows × 8 columns

In [207]:
```
# Check Patient Id not less than 6 digit
count = 0
for i in df['Patient_id'].values:

    if len(str(i))<6:
        df.drop([count], axis=0, inplace=True)
    elif len(str(i))>6:
        df.drop([count], axis=0, inplace=True)
    count = count+1
```

In [208]:
```
# Count the patient_gender types
df['patient_gender'].unique()
```

Out[208]:
```
array(['Female', 'Male'], dtype=object)
```

In [209]:
```
# Export the Data
df.to_csv("patient.csv",index=False)
```

In [210]:
```
# Read the subscriber Details
df = pd.read_csv('subscriber.csv')
```

In [211]:
```
df.head()
```

Out[211]:

	sub _id	first_name	last_name	Street	Birth_date	Gender	Phone	Country	City	Zip Code	Subgrp_id	Elig_ind	eff_date	term_date

Here is another example using subscriber.csv:

In [214]:
```
df.isnull()
```

Out[214]:

	sub _id	first_name	last_name	Street	Birth_date	Gender	Phone	Country	City	Zip Code	Subgrp_id	Elig_ind	eff_date	term_date
0	False	False	False	False	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
95	False	False	False	False	False	False	True	False	False	False	False	False	False	False
96	False	False	False	False	False	False	False	False	False	False	False	False	False	False
97	False	True	False	False	False	False	False	False	False	False	False	False	False	False
98	False	False	False	False	False	False	False	False	False	False	False	False	False	False
99	False	False	False	False	False	False	False	False	False	False	False	False	False	False

100 rows × 14 columns

In [215]:
```
# Count the total Null values
df.isnull().sum()
```

Out[215]:
```
sub _id        0
first_name    27
last_name      0
Street         0
Birth_date     0
Gender         0
Phone          3
Country        0
City           0
Zip Code       0
```

In [216]:
```
# Replace the null values
df["first_name"].fillna("NA", inplace = True)
```

In [217]:
```
# Replace the null values
df["Phone"].fillna("NA", inplace = True)
```

In [218]:
```
# Replace the null values
df["Subgrp_id"].fillna("NA", inplace = True)
```

In [219]:
```
# Replace the null values
df["Elig_ind"].fillna("N", inplace = True)
```

In [220]:
```
df.head(60)
```

Out[220]:

	sub _id	first_name	last_name	Street	Birth_date	Gender	Phone	Country	City	Zip Code	Subgrp_id	Elig_ind	eff_date	ter
0	SUBID10000	Harbir	Vishwakarma	Baria Marg	1924-06-30	Female	+91 0112009318	India	Rourkela	767058	S107	Y	1944-06-30	1
1	SUBID10001	Brahmdev	Sonkar	Lala Marg	1948-12-20	Female	+91 1727749552	India	Tiruvottiyur	34639	S105	Y	1968-12-20	1
2	SUBID10002	Ujjawal	Devi	Mammen Zila	1980-04-16	Male	+91 8547451606	India	Berhampur	914455	S106	N	2000-04-16	2
3	SUBID10003	Ballari	Mishra	Sahni Zila	1969-09-25	Female	+91 0106026841	India	Bihar Sharif	91481	S104	N	1989-09-25	1
4	SUBID10004	Devnath	Srivastav	Magar Zila	1946-05-01	Female	+91 1868774631	India	Bidhannagar	531742	S110	N	1966-05-01	1
5	SUBID10005	Atasi	Seth	Khatri Nagar	1967-10-02	Male	+91 9747336855	India	Amravati	229062	S104	Y	1987-10-02	1
6	SUBID1006	Manish	Maurya	Swaminathan Chowk	1967-06-06	Male	+91 4254294043	India	Panvel	438733	S109	N	1987-06-06	1

In [221]:
```
df.isnull().sum()
```

In [221]:
```
df.isnull().sum()
```

Out[221]:
```
sub _id       0
first_name    0
last_name     0
Street        0
Birth_date    0
Gender        0
Phone         0
Country       0
City          0
Zip Code      0
Subgrp_id     0
Elig_ind      0
eff_date      0
term_date     0
dtype: int64
```

In [222]:
```
# Check the duplicates
df.duplicated()
```

Out[222]:
```
0     False
1     False
2     False
3     False
4     False
      ...
95    False
96    False
97    False
98    False
99    False
Length: 100, dtype: bool
```

In [223]:
```
# Drop duplicates
df.drop_duplicates(subset=None, keep='first', inplace=False, ignore_index=False)
```

Out[223]:

	sub _id	first_name	last_name	Street	Birth_date	Gender	Phone	Country	City	Zip Code	Subgrp_id	Elig_ind	eff_date	term_date
0	SUBID10000	Harbir	Vishwakarma	Baria Marg	1924-06-30	Female	+91 0112009318	India	Rourkela	767058	S107	Y	1944-06-30	1954-01-14
1	SUBID10001	Brahmdev	Sonkar	Lala Marg	1948-12-20	Female	+91 1727749552	India	Tiruvottiyur	34639	S105	Y	1968-12-20	1970-05-16
2	SUBID10002	Ujjawal	Devi	Mammen Zila	1980-04-16	Male	+91 8547451606	India	Berhampur	914455	S106	N	2000-04-16	2008-05-04
3	SUBID10003	Ballari	Mishra	Sahni Zila	1969-09-25	Female	+91 0106026841	India	Bihar Sharif	91481	S104	N	1989-09-25	1995-06-05
4	SUBID10004	Devnath	Srivastav	Magar Zila	1946-05-01	Female	+91 1868774631	India	Bidhannagar	531742	S110	N	1966-05-01	1970-12-09
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
95	SUBID10095	Ekaaksh	Rai	Bansal Ganj	1933-12-02	Others	NA	India	Pimpri-Chinchwad	158186	S107	N	1953-12-02	1960-07-29
96	SUBID10096	Chanak	Sonkar	Kaur	1959-04-07	Others	+91 7284540687	India	Raurkela Industrial Township	899590	S101	Y	1979-04-07	1986-03-07
97	SUBID10097	NA	Sonkar	Rana Ganj	1940-02-04	Others	+91 8908240160	India	Mira-Bhayandar	896586	S107	Y	1960-02-04	1965-01-12
98	SUBID1098	Pushkar	Kumar	Sodhi Zila	1934-10-05	Others	+91 8956368286	India	Korba	910732	S107	Y	1954-10-05	1961-04-05
99	SUBID10099	Shikha	Srivastav	Ahuja Road	1970-09-06	Others	+91 3042509956	India	Nanded	101500	S109	Y	1990-09-06	1997-11-27

100 rows × 14 columns

In [224]:
```
# Check Subscriber Id not less than 9 digit
count = 0
for i in df['sub _id'].values:

    if len(i)<9:
        df.drop([count], axis=0, inplace=True)
    elif len(str(i))>10:
        df.drop([count], axis=0, inplace=True)
    count = count+1
```

In [225]:
```
# Check the Elig_ind types
df['Elig_ind'].unique()
```

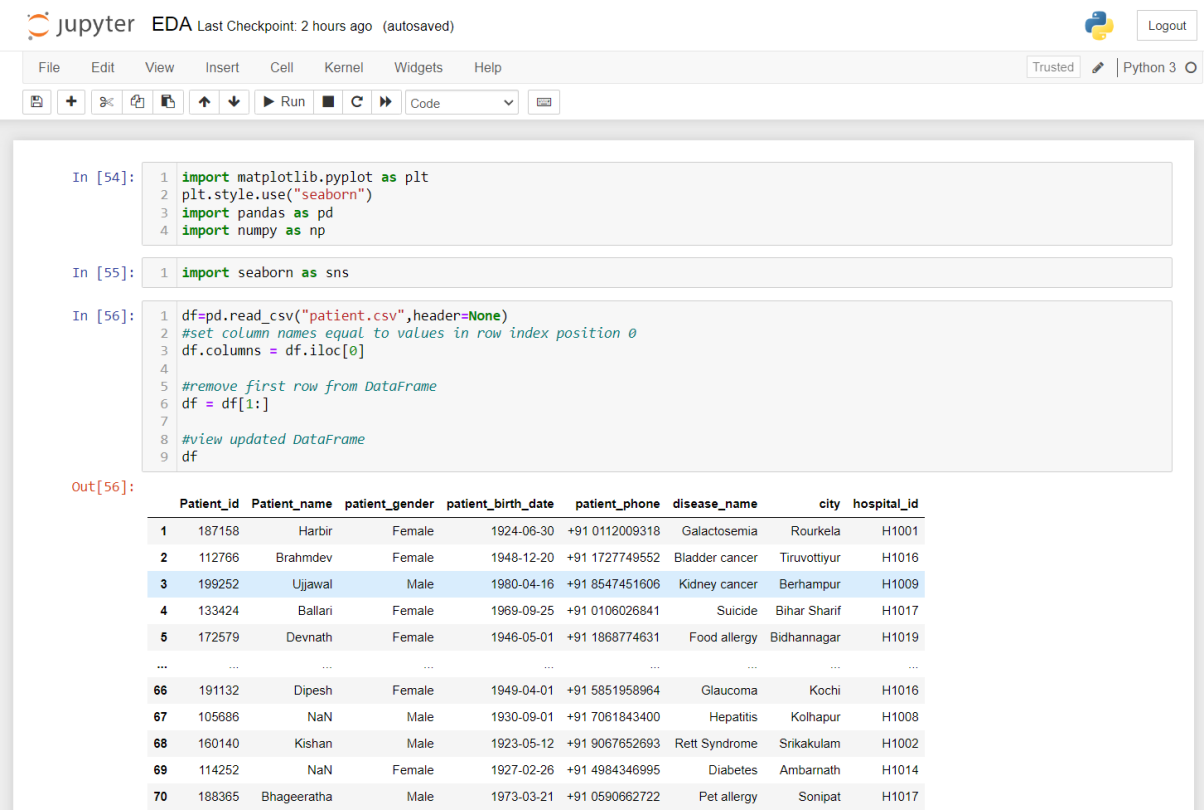
Out[225]:
```
array(['Y', 'N'], dtype=object)
```

In [226]:
```
# Check always eff_date is greater than term_date
count = 0
for x,y in df[['eff_date','term_date']].values:
    dob1 = datetime.strptime(x,'%Y-%m-%d').date()
    dob2 = datetime.strptime(y,'%Y-%m-%d').date()
    if dob1 > dob2:
        df.drop([count], axis=0, inplace=True)
    count = count + 1
```

In [227]:
```
# Export the Data
df.to_csv('subscriber.csv',index=False)
```

Exploratory Data Analysis (EDA):

EDA is used to analyze and investigate data sets and summarize their main characteristics.



In [54]:
```
import matplotlib.pyplot as plt
plt.style.use("seaborn")
import pandas as pd
import numpy as np
```

In [55]:
```
import seaborn as sns
```

In [56]:
```
df=pd.read_csv("patient.csv",header=None)
#set column names equal to values in row index position 0
df.columns = df.iloc[0]

#remove first row from DataFrame
df = df[1:]

#view updated DataFrame
df
```

Out[56]:

	Patient_id	Patient_name	patient_gender	patient_birth_date	patient_phone	disease_name	city	hospital_id
1	187158	Harbir	Female	1924-06-30	+91 0112009318	Galactosemia	Rourkela	H1001
2	112766	Brahmdev	Female	1948-12-20	+91 1727749552	Bladder cancer	Tiruvottiyur	H1016
3	199252	Ujjawal	Male	1980-04-16	+91 8547451606	Kidney cancer	Berhampur	H1009
4	133424	Ballari	Female	1969-09-25	+91 0106026841	Suicide	Bihar Sharif	H1017
5	172579	Devnath	Female	1946-05-01	+91 1868774631	Food allergy	Bidhannagar	H1019
...	...	...	...	...	...	...	...	...
66	191132	Dipesh	Female	1949-04-01	+91 5851958964	Glaucoma	Kochi	H1016
67	105686	NaN	Male	1930-09-01	+91 7061843400	Hepatitis	Kolhapur	H1008
68	160140	Kishan	Male	1923-05-12	+91 9067652693	Rett Syndrome	Srikakulam	H1002
69	114252	NaN	Female	1927-02-26	+91 4984346995	Diabetes	Ambarnath	H1014
70	188365	Bhageeratha	Male	1973-03-21	+91 0590662722	Pet allergy	Sonipat	H1017

In [57]:
```
#reset index values
df.reset_index(drop=True, inplace=True)

#view updated DataFrame
df
```

Out[57]:

	Patient_id	Patient_name	patient_gender	patient_birth_date	patient_phone	disease_name	city	hospital_id
0	187158	Harbir	Female	1924-06-30	+91 0112009318	Galactosemia	Rourkela	H1001
1	112766	Brahmdev	Female	1948-12-20	+91 1727749552	Bladder cancer	Tiruvottiyur	H1016
2	199252	Ujjawal	Male	1980-04-16	+91 8547451606	Kidney cancer	Berhampur	H1009
3	133424	Ballari	Female	1969-09-25	+91 0106026841	Suicide	Bihar Sharif	H1017
4	172579	Devnath	Female	1946-05-01	+91 1868774631	Food allergy	Bidhannagar	H1019
...	...	...	...	...	...	...	...	...
65	191132	Dipesh	Female	1949-04-01	+91 5851958964	Glaucoma	Kochi	H1016
66	105686	NaN	Male	1930-09-01	+91 7061843400	Hepatitis	Kolhapur	H1008
67	160140	Kishan	Male	1923-05-12	+91 9067652693	Rett Syndrome	Srikakulam	H1002
68	114252	NaN	Female	1927-02-26	+91 4984346995	Diabetes	Ambarnath	H1014
69	188365	Bhageeratha	Male	1973-03-21	+91 0590662722	Pet allergy	Sonipat	H1017

70 rows × 8 columns

In [58]:
```
def age(born):
    born = datetime.strptime(born, "%Y-%m-%d").date()
    today = date.today()
    return today.year - born.year - ((today.month,
                                      today.day) < (born.month,
                                                    born.day))
```

A column called “Age” is calculated and added to the “patient.csv” dataset.

In [59]:
```
from datetime import datetime, date
```

In [60]:
```
df['Age'] = df['patient_birth_date'].apply(age)
```

In [61]:
```
df
```

Out[61]:

	Patient_id	Patient_name	patient_gender	patient_birth_date	patient_phone	disease_name	city	hospital_id	Age
0	187158	Harbir	Female	1924-06-30	+91 0112009318	Galactosemia	Rourkela	H1001	98
1	112766	Brahmdev	Female	1948-12-20	+91 1727749552	Bladder cancer	Tiruvottiyur	H1016	74
2	199252	Ujjawal	Male	1980-04-16	+91 8547451606	Kidney cancer	Berhampur	H1009	42
3	133424	Ballari	Female	1969-09-25	+91 0106026841	Suicide	Bihar Sharif	H1017	53
4	172579	Devnath	Female	1946-05-01	+91 1868774631	Food allergy	Bidhannagar	H1019	76
...	...	...	...	...	...	...	...	...	...
65	191132	Dipesh	Female	1949-04-01	+91 5851958964	Glaucoma	Kochi	H1016	73
66	105686	NaN	Male	1930-09-01	+91 7061843400	Hepatitis	Kolhapur	H1008	92
67	160140	Kishan	Male	1923-05-12	+91 9067652693	Rett Syndrome	Srikakulam	H1002	99
68	114252	NaN	Female	1927-02-26	+91 4984346995	Diabetes	Ambarnath	H1014	96
69	188365	Bhageeratha	Male	1973-03-21	+91 0590662722	Pet allergy	Sonipat	H1017	49

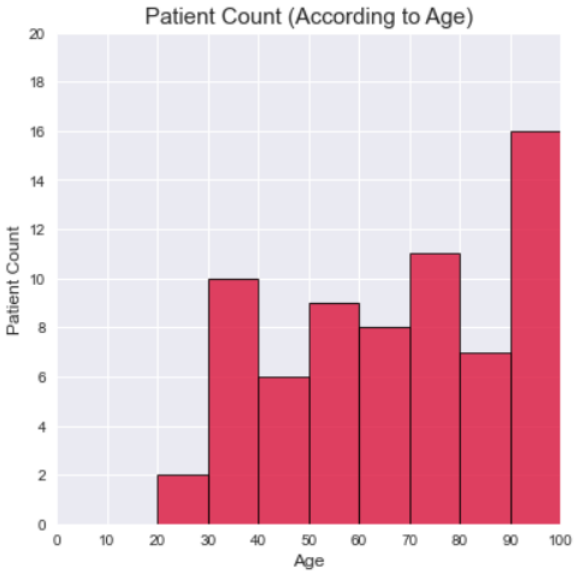
70 rows × 9 columns

In [62]:
```
df['Age']
```

Out[62]:
```
0    98
1    74
2    42
3    53
4    76
     ..
```

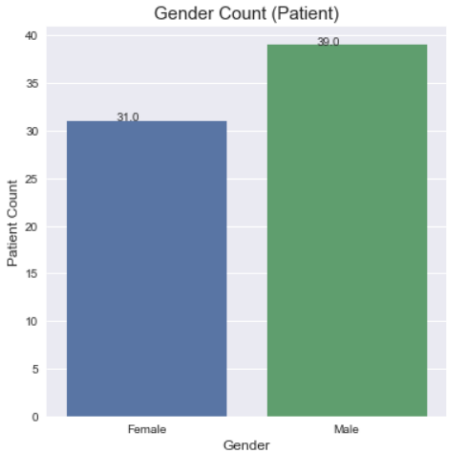
A graph of Patient Count vs Age is plotted.

In [63]:
```
plt.figure(figsize=(6,6))
bins=[0,10,20,30,40,50,60,70,80,90,100,110]
a=df['Age']
plt.hist(a, bins, facecolor='crimson', alpha=0.8, edgecolor='k', linewidth=1)
plt.xlabel("Age",fontsize=12)
plt.ylabel("Patient Count",fontsize=12)
plt.locator_params('y', nbins=12)
plt.ylim(ymin=0,ymax=20)
plt.locator_params('x', nbins=10)
plt.xlim(xmin=0,xmax=100)
plt.title("Patient Count (According to Age)",fontsize=15)
plt.savefig('PatientCount(Age).png')
```



Similarly for Patient Count vs Gender

In [64]:
```
plt.figure(figsize=(6,6))
ax=sns.countplot(x='patient_gender',data=df)
ax.set_ylabel("Patient Count", fontsize = 12)
ax.set_xlabel("Gender", fontsize = 12)
ax.set_title("Gender Count (Patient)", fontsize=15)
for p in ax.patches:
    ax.annotate('{:.1f}'.format(p.get_height()), (p.get_x()+0.25, p.get_height()+0.01))
plt.savefig('GenderCount(Patient).png')
```



Similarly the above steps are done for other datasets:

Here is another example using the claims.json dataset

In [65]:
```
1 js=pd.read_json('claims.json')
```

In [66]:
```
1 js
```

Out[66]:

	claim_id	patient_id	disease_name	SUB_ID	Claim_Or_Rejected	claim_type	claim_amount	claim_date
0	0	187158	Galactosemia	SUBID1000	N	claims of value	79874	1949-03-14
1	1	112766	Bladder cancer	SUBID10001	N	claims of policy	151142	1970-03-16
2	2	199252	Kidney cancer	SUBID10002	N	claims of value	59924	2008-02-03
3	3	133424	Suicide	SUBID10003	N	claims of fact	143120	1995-02-08
4	4	172579	Food allergy	SUBID10004	Y	claims of value	168634	1967-05-23
...	...	...	...	...	...	...	...	...
65	65	191132	Glaucoma	SUBID1065	Y	claims of policy	81980	1969-05-31
66	66	105686	Hepatitis	SUBID10066	N	claims of fact	13667	1957-09-12
67	67	160140	Rett Syndrome	SUBID1067	N	claims of value	109433	1944-12-25
68	68	114252	Diabetes	SUBID10068	N	claims of policy	152901	1948-02-13
69	69	188365	Pet allergy	SUBID10069	N	claims of fact	99313	1994-08-25

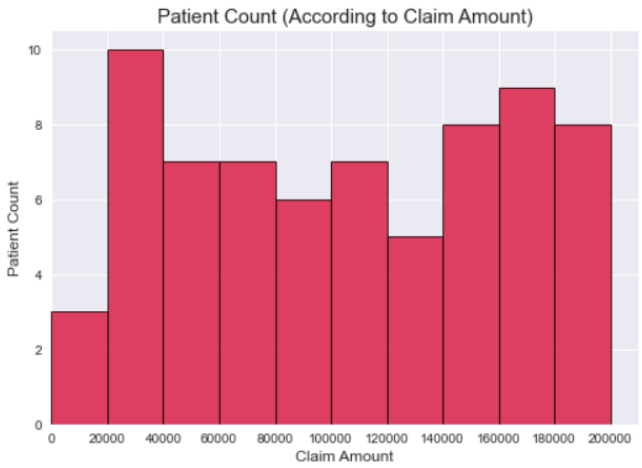
70 rows × 8 columns

In [67]:
```
1 plt.figure(figsize=(5,5))
2 ax=sns.countplot(x='Claim_Or_Rejected',data=js,hue='Claim_Or_Rejected')
3 ax.set_ylabel("Count",fontsize=12)
4 ax.set_xlabel("Status", fontsize=12)
5 ax.set_title("Claim Status",fontsize=15)
6 for p in ax.patches:
7     ax.annotate('{:.1f}'.format(p.get_height()), (p.get_x()+0.25, p.get_height()+0.01))
8 plt.savefig('ClaimStatus.png')
```

70 rows × 8 columns

In [67]:
```
1 plt.figure(figsize=(5,5))
2 ax=sns.countplot(x='Claim_Or_Rejected',data=js,hue='Claim_Or_Rejected')
3 ax.set_ylabel("Count",fontsize=12)
4 ax.set_xlabel("Status", fontsize=12)
5 ax.set_title("Claim Status",fontsize=15)
6 for p in ax.patches:
7     ax.annotate('{:.1f}'.format(p.get_height()), (p.get_x()+0.25, p.get_height()+0.01))
8 plt.savefig('ClaimStatus.png')
```

In [68]:
```
bins=[0,20000,40000,60000,80000,100000,120000,140000,160000,180000,200000,220000,240000]
a=js['claim_amount']
plt.hist(a, bins, facecolor='crimson', alpha=0.8, edgecolor='k', linewidth=1)
plt.xlabel("Claim Amount",fontsize=12)
plt.ylabel("Patient Count",fontsize=12)
plt.locator_params('x', nbins=12)
plt.xlim(xmin=0,xmax=210000)
plt.title("Patient Count (According to Claim Amount)",fontsize=15)
plt.savefig('Patient_count(claimAmt).png')
```



5.3 STORAGE, RETRIEVAL AND MANAGEMENT OF DATA:

Connecting to MySQL using Python :

In [33]:
```
# Importing data preprocessing lib
import pandas as pd

```

In [34]:
```
pip install mysql-connector-python-rf
```

```
Requirement already satisfied: mysql-connector-python-rf in c:\users\sheeba\anaconda3\lib\site-packages (2.2.2)
Note: you may need to restart the kernel to use updated packages.
```

In [35]:
```
# Import mysql connector and stablish connection
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",user="root",password="password")

print(mydb)
```

```
<mysql.connector.connection.MySQLConnection object at 0x0000022220892EE0>
```

Viewing Existing Databases:

In [36]:
```
# Print the databases (list of databses)
mycursor = mydb.cursor()

mycursor.execute("show databases")
myresult = mycursor.fetchall()

for x in myresult:
    print(x)
```

```
('a',)
('database1',)
('demo',)
('epes_db',)
('healthcare',)
('information_schema',)
('loginregister',)
('mysql',)
('performance_schema',)
('phpmyadmin',)
('registration',)
('test',)
('testing',)
('userregister',)
('userregistration',)
```

Creating and using new database(health_care):

In [38]:
```
# Create new database

mycursor.execute("CREATE DATABASE HEALTH_CARE")
```

In [40]:
```
# Use new database

mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  password="password",
    database="HEALTH_CARE"
)

mycursor = mydb.cursor()
```

Creating new tables:

In [41]:
```
# Create new table groups

mycursor.execute("""CREATE TABLE groups_tble (grp_sk int NOT NULL UNIQUE AUTO_INCREMENT,grp_id VARCHAR(6) NOT NULL PRIMARY KE
                grp_name VARCHAR(90),premium_written int NOT NULL,city VARCHAR(20),
                zip_code int,country VARCHAR(5),grp_type VARCHAR(10))""")
```

In [42]:
```
# Create new table subgroup
mycursor.execute("""CREATE TABLE subgroup (subgrp_sk int NOT NULL UNIQUE AUTO_INCREMENT,subgrp_id VARCHAR(4) NOT NULL PRIMAR
                subgrp_name VARCHAR(90),monthly_premium float(6,2))""")
```

In [43]:
```
# Create new table group_subgroup
mycursor.execute("""CREATE TABLE group_subgroup (grpsub_sk int NOT NULL UNIQUE AUTO_INCREMENT,
                    g_id VARCHAR(6) NOT NULL,
                    s_id VARCHAR(4) NOT NULL,
                    FOREIGN KEY(g_id) REFERENCES groups_tble(grp_id),
                    FOREIGN KEY(s_id) REFERENCES subgroup(subgrp_id))""")
```

In [44]:
```
# Create new table disease
mycursor.execute("""CREATE TABLE disease (disease_id int NOT NULL PRIMARY KEY ,disease_name VARCHAR(30) NOT NULL ,
                subgrp_id VARCHAR(4),FOREIGN KEY(subgrp_id) REFERENCES subgroup(subgrp_id))""")
```

In [45]:
```
# Create new table subscriber
mycursor.execute("""CREATE TABLE subscriber (sub_id VARCHAR(10) NOT NULL PRIMARY KEY ,first_name VARCHAR(30) ,
                last_name VARCHAR(20),street VARCHAR(30),birth_date DATE,gender VARCHAR(6),phone VARCHAR(15),city VARCHAR(3
                zip_code int,country VARCHAR(10),subgrp_id VARCHAR(4),elig_ind VARCHAR(2) NOT NULL,eff_date DATE NOT NULL,t
```

In [46]:
```
# create new table hospital_details
mycursor.execute("""CREATE TABLE hospital_details (hospital_id VARCHAR(5) NOT NULL PRIMARY KEY ,hospital_name VARCHAR(255),
                city VARCHAR(20),state VARCHAR(20),country VARCHAR(6))""")

```

In [47]:
```
# Create new table patient_details
mycursor.execute("""CREATE TABLE patient_details(patient_id int NOT NULL PRIMARY KEY,patient_name VARCHAR(20),patient_gender
```

In [48]:
```
# Create new table claims
mycursor.execute("""CREATE TABLE claims (claim_id int NOT NULL AUTO_INCREMENT PRIMARY KEY ,patient_id int NOT NULL,disease_n
                sub_id VARCHAR(10),claim_or_rejected VARCHAR(5),claim_type VARCHAR(20),claim_amount float(8,2),claim_date D
```

Inserting values into tables from preprocessed data files:

In [51]:
```
#inserting data into table group
with open('group.csv','r') as file:
    mycursor.execute('''use HEALTH_CARE''')
    data=file.readlines()
    for row in data:
            list=row.split(",")
            top=tuple([list[0],int(list[1]),int(list[2]),list[3],list[4],list[5],list[6]])
            print(top)
            sql = "INSERT INTO HEALTH_CARE.groups_tble(country,premium_written,zip_code,grp_id,grp_name,grp_type,city) VALUE
            mycursor.execute(sql, top)
            mydb.commit()
```

```
('0', 1, 2, '3', '4', '5', '6')
('India', 72000, 482018, 'GRP101', 'Life Insurance Corporation of India', 'Govt.', 'Mumbai')
('India', 45000, 482049, 'GRP102', 'HDFC Standard Life Insurance Co. Ltd.', 'Private', 'Mumbai')
('India', 64000, 482030, 'GRP103', 'Max Life Insurance Co. Ltd.', 'Private', 'Delhi')
('India', 59000, 482028, 'GRP104', 'ICICI Prudential Life Insurance Co. Ltd.', 'Private', 'Mumbai')
('India', 37000, 482014, 'GRP105', 'Kotak Mahindra Life Insurance Co. Ltd.', 'Private', 'Mumbai')
('India', 89000, 482011, 'GRP106', 'Aditya Birla Sun Life Insurance Co. Ltd.', 'Private', 'Mumbai')
('India', 70000, 482006, 'GRP107', 'TATA AIG Life Insurance Co. Ltd.', 'Private', 'Mumbai')
('India', 52000, 482034, 'GRP108', 'SBI Life Insurance Co. Ltd.', 'Private', 'Mumbai')
('India', 78000, 482032, 'GRP109', 'Exide Life Insurance Co. Ltd.', 'Private', 'Bangalore')
('India', 48000, 482015, 'GRP110', 'Bajaj Allianz Life Insurance Co. Ltd.', 'Private', 'Pune')
('India', 57000, 482011, 'GRP111', 'PNB MetLife India Insurance Co. Ltd.', 'Private', 'Mumbai')
('India', 57000, 482022, 'GRP112', 'Reliance Nippon Life Insurance Company', 'Private', 'Mumbai')
('India', 64000, 482009, 'GRP113', 'Aviva Life Insurance Company India Ltd.', 'Private', 'Gurugram')
('India', 33000, 482043, 'GRP114', 'Sahara India Life Insurance Co. Ltd.', 'Private', 'Lucknow')
('India', 79000, 482036, 'GRP115', 'Shriram Life Insurance Co. Ltd.', 'Private', 'Hyderabad')
('India', 32000, 482002, 'GRP116', 'Bharti AXA Life Insurance Co. Ltd.', 'Private', 'Mumbai')
('India', 59000, 482017, 'GRP117', 'Future Generali India Life Insurance Co. Ltd.', 'Private', 'Mumbai')
('India', 97000, 482023, 'GRP118', 'IDBI Federal Life Insurance Co. Ltd.', 'Private', 'Mumbai')
('India', 47000, 482046, 'GRP119', 'Canara HSBC Oriental Bank of Commerce Life Insurance Co. Ltd.', 'Private', 'Gurugram')
```

In [52]:

```
#insert data into subgroup
with open('subgroup.csv','r') as file:
    mycursor.execute('''use HEALTH_CARE''')
    data=file.readlines()
    for row in data:
        list=row.split(",")
        top=tuple([list[0],list[1],float(list[2])])
        print(top)
        sql = "INSERT INTO HEALTH_CARE.subgroup(subgrp_id,subgrp_name,monthly_premium) VALUES(%s,%s,%s)"
        mycursor.execute(sql, top)
        mydb.commit()
```

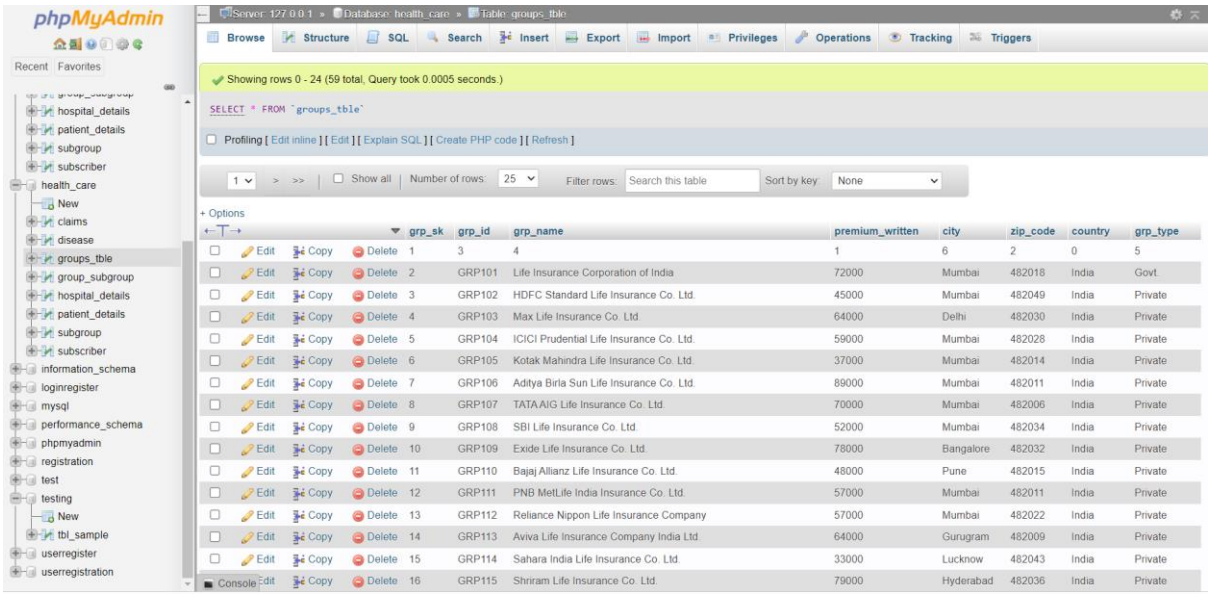
```
('0', '1', 2.0)
('S101', 'Deficiency Diseases', 3000.0)
('S102', 'Accident', 1000.0)
('S103', 'Physiology', 2000.0)
('S104', 'Therapy', 1500.0)
('S105', 'Allergies', 2300.0)
('S106', 'Self inflicted', 1200.0)
('S107', 'Cancer', 3200.0)
('S108', 'Infectious disease', 1500.0)
('S109', 'Hereditary', 2000.0)
('S110', 'Viral', 1000.0)
```

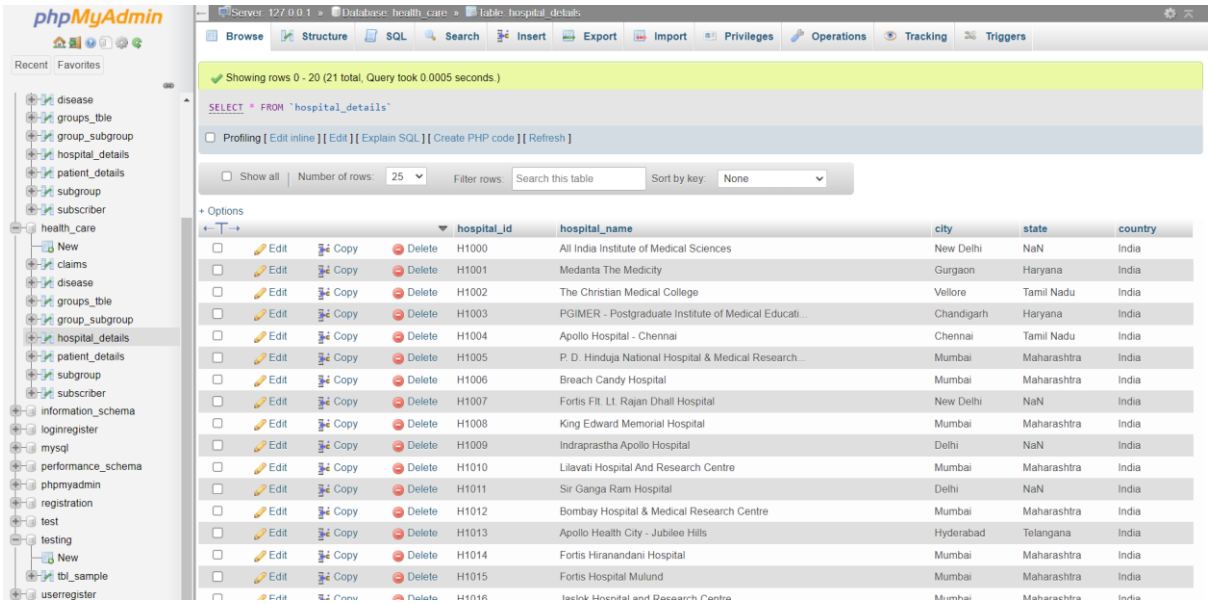
In [54]:

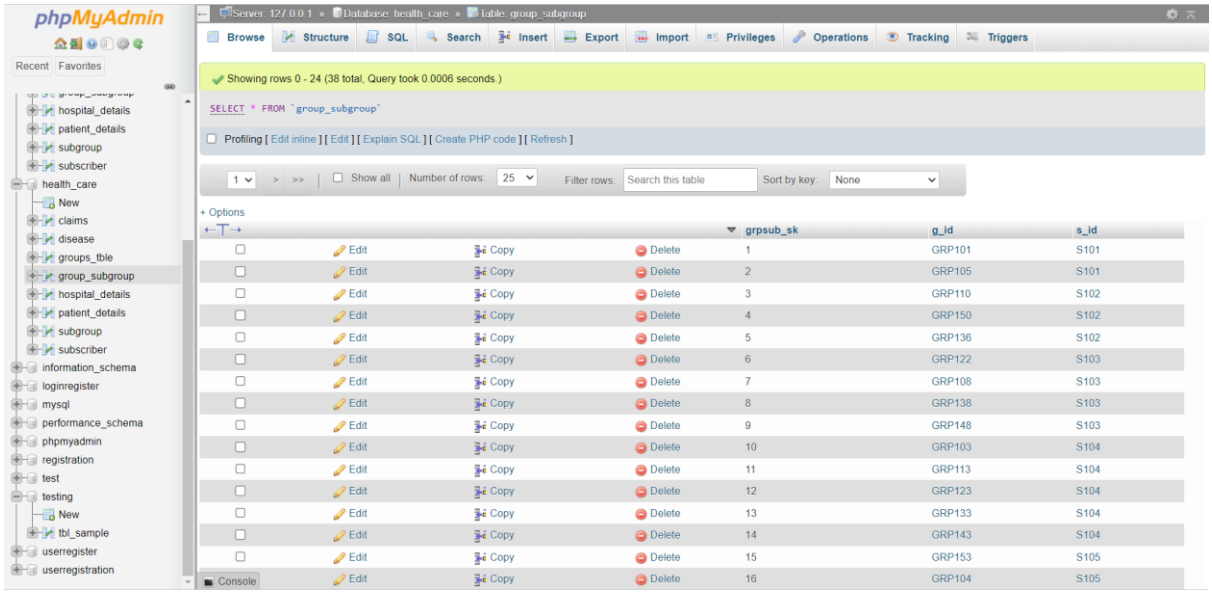
```
#inserting data into hospital_details
with open('hospital.csv','r') as file:
    mycursor.execute('''use HEALTH_CARE''')
    data=file.readlines()
    for row in data:
        list=row.split(",")
        top=tuple(list)
        print(top)
        sql = "INSERT INTO hospital_details(hospital_id,hospital_name,city,state,country) VALUES(%s,%s,%s,%s,%s)"
        mycursor.execute(sql,top)
        mydb.commit()
```

```
('Hospital_id', 'Hospital_name', 'city', 'state', 'country\n')
('H1000', 'All India Institute of Medical Sciences', 'New Delhi', 'NaN', 'India\n')
('H1001', 'Medanta The Medicity', 'Gurgaon', 'Haryana', 'India\n')
('H1002', 'The Christian Medical College', 'Vellore', 'Tamil Nadu', 'India\n')
('H1003', 'PGIMER - Postgraduate Institute of Medical Education and Research', 'Chandigarh', 'Haryana', 'India\n')
('H1004', 'Apollo Hospital - Chennai', 'Chennai', 'Tamil Nadu', 'India\n')
('H1005', 'P. D. Hinduja National Hospital & Medical Research Centre', 'Mumbai', 'Maharashtra', 'India\n')
('H1006', 'Breach Candy Hospital', 'Mumbai', 'Maharashtra', 'India\n')
('H1007', 'Fortis Flt. Lt. Rajan Dhall Hospital', 'New Delhi', 'NaN', 'India\n')
('H1008', 'King Edward Memorial Hospital', 'Mumbai', 'Maharashtra', 'India\n')
('H1009', 'Indraprastha Apollo Hospital', 'Delhi', 'NaN', 'India\n')
('H1010', 'Lilavati Hospital And Research Centre', 'Mumbai', 'Maharashtra', 'India\n')
('H1011', 'Sir Ganga Ram Hospital', 'Delhi', 'NaN', 'India\n')
('H1012', 'Bombay Hospital & Medical Research Centre', 'Mumbai', 'Maharashtra', 'India\n')
('H1013', 'Apollo Health City - Jubilee Hills', 'Hyderabad', 'Telangana', 'India\n')
('H1014', 'Fortis Hiranandani Hospital', 'Mumbai', 'Maharashtra', 'India\n')
('H1015', 'Fortis Hospital Mulund', 'Mumbai', 'Maharashtra', 'India\n')
('H1016', 'Jaslok Hospital and Research Centre', 'Mumbai', 'Maharashtra', 'India\n')
('H1017', 'Manipal Hospitals', 'Bengaluru', 'Karnataka', 'India\n')
('H1018', 'Yashoda Hospital Secunderabad', 'Hyderabad', 'Telangana', 'India\n')
('H1019', 'Apollo Hospitals - Bannerghatta Road', 'Bengaluru', 'Karnataka', 'India\n')
```

Viewing the database and tables in phpMyAdmin:







5.4 DATA INGESTION SERVICES:

Sqoop version 1.4.4 is installed

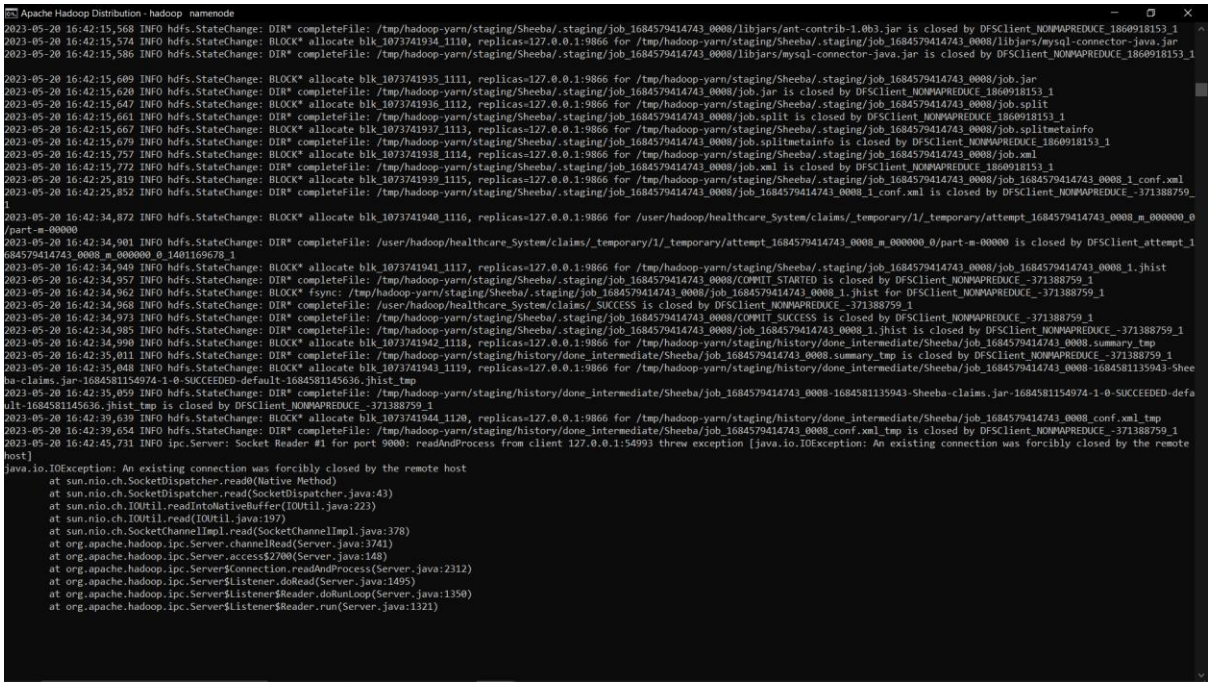
```
C:\WINDOWS\system32>sqoop version
Warning: HBASE_HOME and HBASE_VERSION not set.
Warning: HCAT_HOME not set
Warning: HCATALOG_HOME does not exist HCatalog imports will fail.
Please set HCATALOG_HOME to the root of your HCatalog installation.
Warning: ACCUMULO_HOME not set.
Warning: ZOOKEEPER_HOME not set.
Warning: HBASE_HOME does not exist HBase imports will fail.
Please set HBASE_HOME to the root of your HBase installation.
Warning: ACCUMULO_HOME does not exist Accumulo imports will fail.
Please set ACCUMULO_HOME to the root of your Accumulo installation.
Warning: ZOOKEEPER_HOME does not exist Accumulo imports will fail.
Please set ZOOKEEPER_HOME to the root of your Zookeeper installation.
Sqoop 1.4.4
git commit id 050a2015514533bc25f3134a33401470ee9353ad
Compiled by jarcec on Wed Jul 31 08:37:07 PDT 2013

C:\WINDOWS\system32>
```

Hadoop and Hive are also Installed and configured.

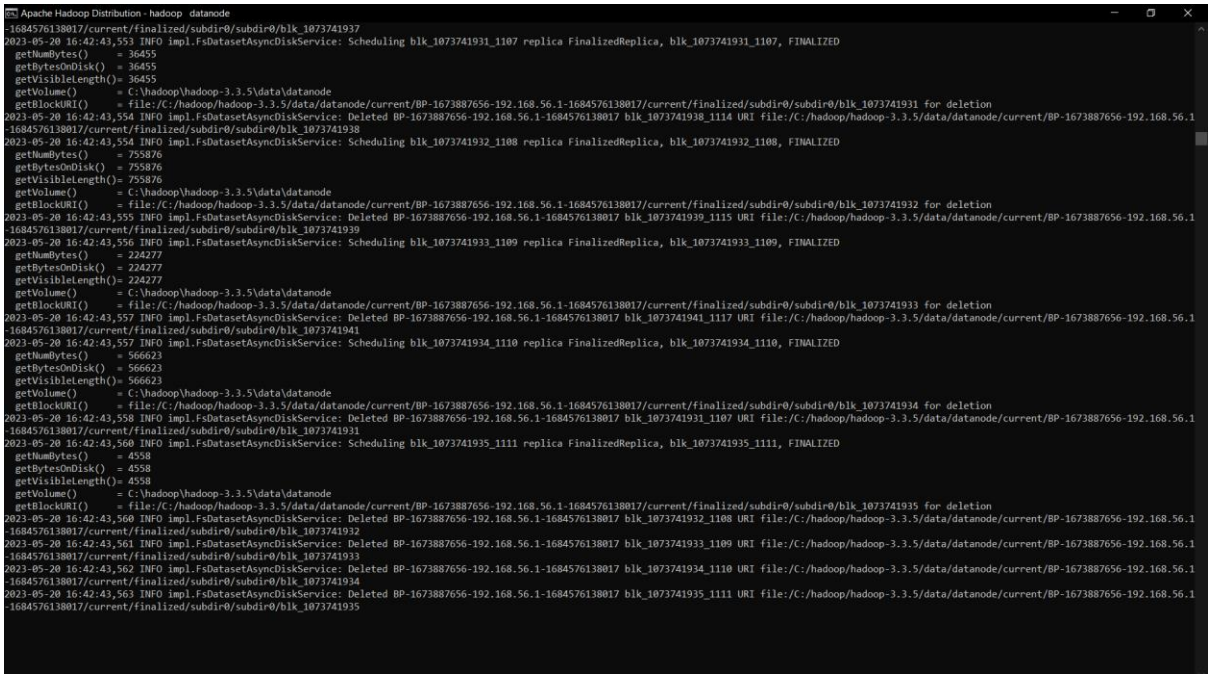
start-all.cmd is given in sbin directory of hadoop

```
C:\hadoop\hadoop-3.3.5\sbin>start-all.cmd
This script is Deprecated. Instead use start-dfs.cmd and start-yarn.cmd
starting yarn daemons

C:\hadoop\hadoop-3.3.5\sbin>
```

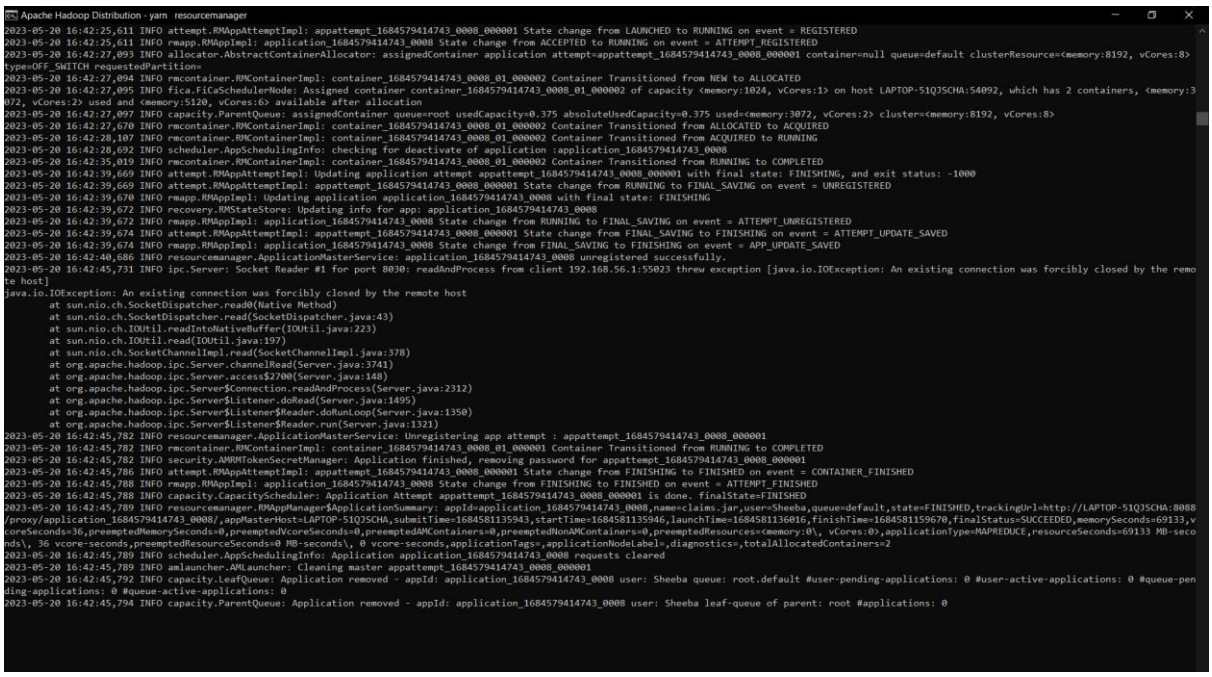
The Hadoop namenode starts



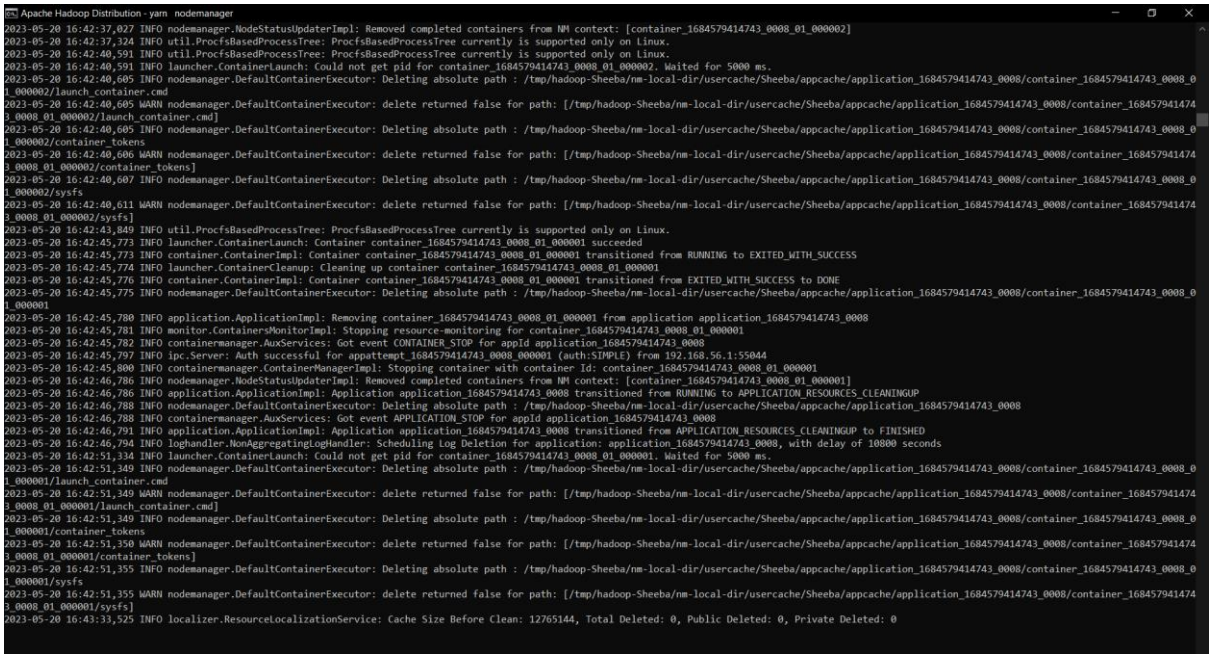
The Hadoop datanode starts



The Hadoop Resourcemanager starts

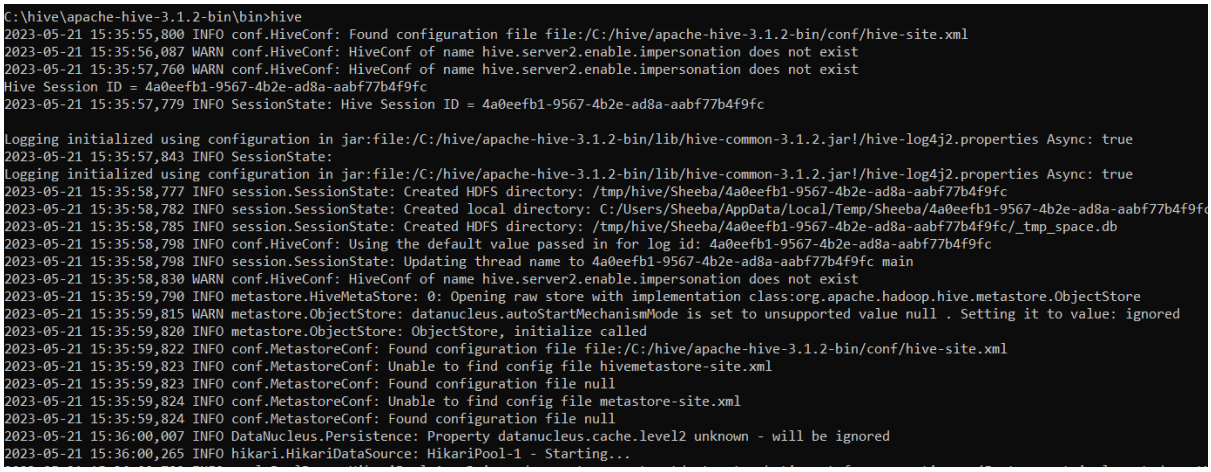


The Hadoop Nodemanager starts



Derby is installed and started

Hive is started



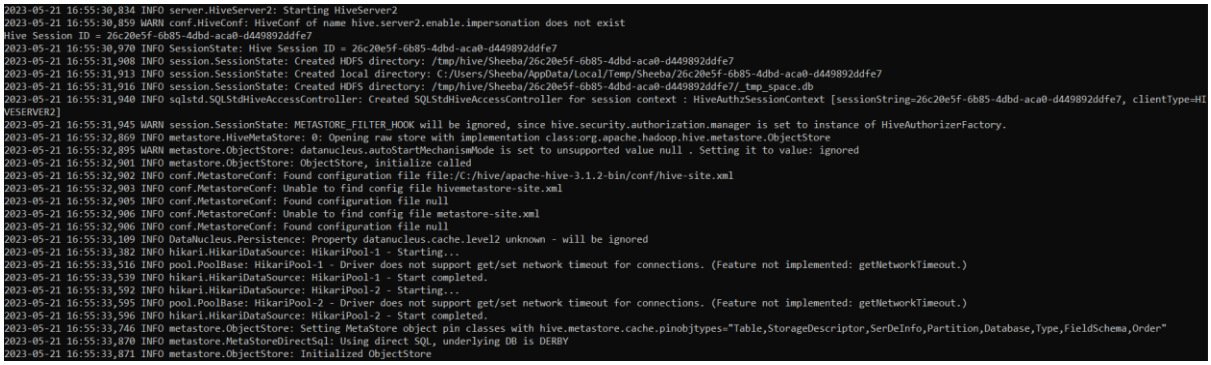
Schematool is initialized

```
Sheeba@LAPTOP-51QJSCHA /cygdrive/c/hive/apache-hive-3.1.2-bin/bin
$ schematool -dbType derby -initSchema
WARNING: DEFAULT_LIBEXEC_DIR ignored. It has been replaced by HADOOP_DEFAULT_LIBEXEC_DIR.
Metastore connection URL:        jdbc:derby://localhost:1527/metastore_db;create=true
Metastore Connection Driver :    org.apache.derby.jdbc.ClientDriver
Metastore connection User:       APP
Starting metastore schema initialization to 3.1.0
Initialization script hive-schema-3.1.0.derby.sql
```

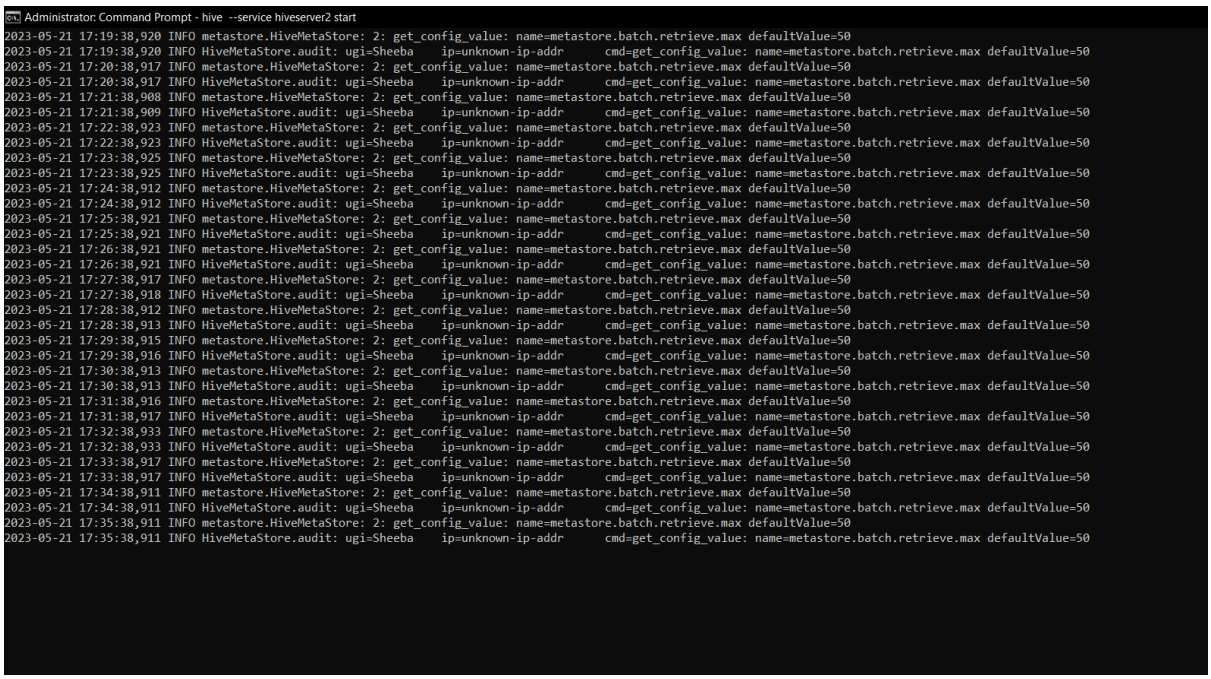
```
Initialization script completed
schemaTool completed

Sheeba@LAPTOP-51QJSCHA /cygdrive/c/hive/apache-hive-3.1.2-bin/bin
```

hive server2 starts

```
Administrator: Command Prompt - hive  --service hiveserver2 start
Microsoft Windows [Version 10.0.19045.2965]
(c) Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>hive --service hiveserver2 start
2023-05-21 16:55:28,909 INFO conf.HiveConf: Found configuration file file:/C:/hive/apache-hive-3.1.2-bin/conf/hive-site.xml
2023-05-21 16:55:29,287 WARN conf.HiveConf: HiveConf of name hive.server2.enable.impersonation does not exist
2023-05-21 16:55:30,802 INFO server.HiveServer2: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting HiveServer2
STARTUP_MSG:   host = LAPTOP-51QJSCHA/192.168.56.1
STARTUP_MSG:   args = [start]
STARTUP_MSG:   version = 3.1.2
```

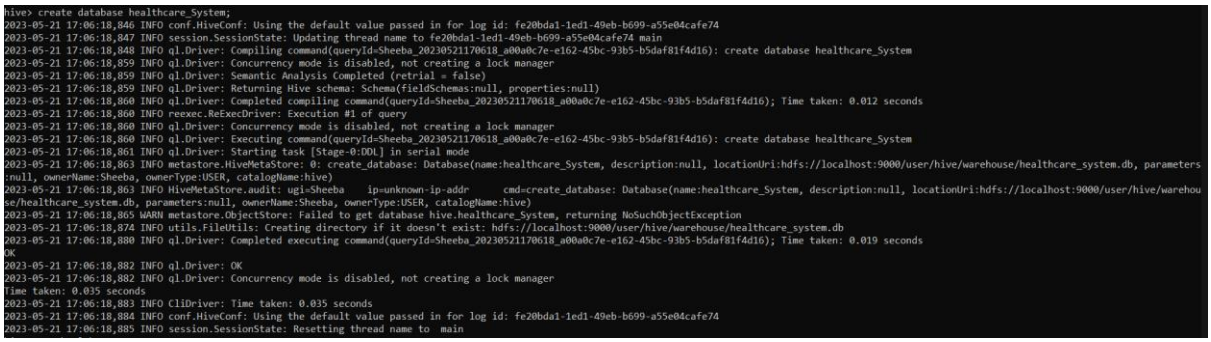


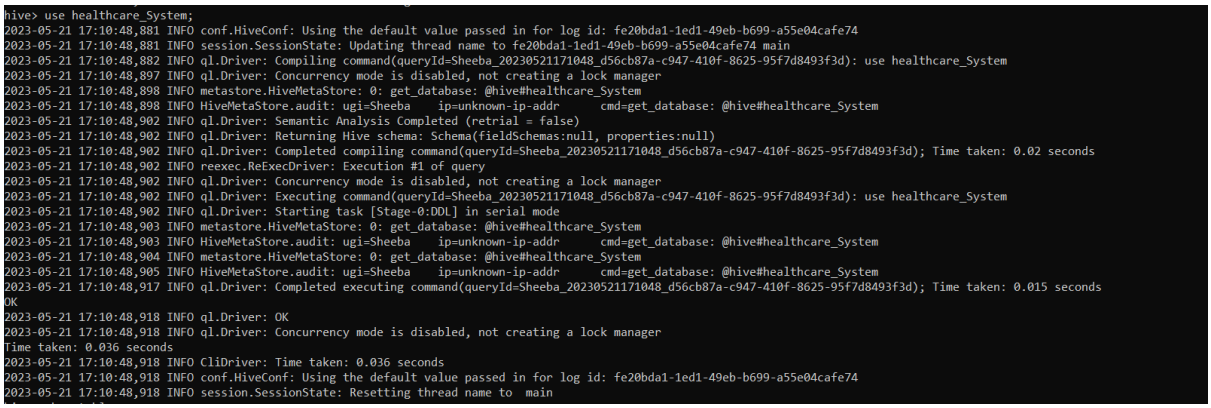
```
2023-05-21 16:55:30,834 INFO server.HiveServer2: Starting HiveServer2
2023-05-21 16:55:30,859 WARN conf.HiveConf: HiveConf of name hive.server2.enable.impersonation does not exist
Hive Session ID = 26c20e5f-6b85-4dbd-aca0-d449892ddfe7
2023-05-21 16:55:30,970 INFO SessionState: Hive Session ID = 26c20e5f-6b85-4dbd-aca0-d449892ddfe7
2023-05-21 16:55:31,908 INFO session.SessionState: Created HDFS directory: /tmp/hive/Sheeba/26c20e5f-6b85-4dbd-aca0-d449892ddfe7
2023-05-21 16:55:31,913 INFO session.SessionState: Created local directory: C:/Users/Sheeba/AppData/Local/Temp/Sheeba/26c20e5f-6b85-4dbd-aca0-d449892ddfe7
2023-05-21 16:55:31,916 INFO session.SessionState: Created HDFS directory: /tmp/hive/Sheeba/26c20e5f-6b85-4dbd-aca0-d449892ddfe7/_tmp_space.db
2023-05-21 16:55:31,940 INFO sqlstd.SQLStdHiveAccessController: Created SQLStdHiveAccessController for session context : HiveAuthzSessionContext [sessionString=26c20e5f-6b85-4dbd-aca0-d449892ddfe7, clientType=HI
VESERVER2]
2023-05-21 16:55:31,945 WARN session.SessionState: METASTORE_FILTER_HOOK will be ignored, since hive.security.authorization.manager is set to instance of HiveAuthorizerFactory.
2023-05-21 16:55:32,869 INFO metastore.HiveMetaStore: 0: Opening raw store with implementation class:org.apache.hadoop.hive.metastore.ObjectStore
2023-05-21 16:55:32,895 WARN metastore.ObjectStore: datanucleus.autoStartMechanismMode is set to unsupported value null . Setting it to value: ignored
2023-05-21 16:55:32,901 INFO metastore.ObjectStore: ObjectStore, initialize called
2023-05-21 16:55:32,902 INFO conf.MetastoreConf: Found configuration file file:/C:/hive/apache-hive-3.1.2-bin/conf/hive-site.xml
2023-05-21 16:55:32,903 INFO conf.MetastoreConf: Unable to find config file hivemetastore-site.xml
2023-05-21 16:55:32,905 INFO conf.MetastoreConf: Found configuration file null
2023-05-21 16:55:32,906 INFO conf.MetastoreConf: Unable to find config file metastore-site.xml
2023-05-21 16:55:32,906 INFO conf.MetastoreConf: Found configuration file null
2023-05-21 16:55:33,109 INFO DataNucleus.Persistence: Property datanucleus.cache.level2 unknown - will be ignored
2023-05-21 16:55:33,382 INFO hikari.HikariDataSource: HikariPool-1 - Starting...
2023-05-21 16:55:33,516 INFO pool.PoolBase: HikariPool-1 - Driver does not support get/set network timeout for connections. (Feature not implemented: getNetworkTimeout.)
2023-05-21 16:55:33,539 INFO hikari.HikariDataSource: HikariPool-1 - Start completed.
2023-05-21 16:55:33,592 INFO hikari.HikariDataSource: HikariPool-2 - Starting...
2023-05-21 16:55:33,595 INFO pool.PoolBase: HikariPool-2 - Driver does not support get/set network timeout for connections. (Feature not implemented: getNetworkTimeout.)
2023-05-21 16:55:33,596 INFO hikari.HikariDataSource: HikariPool-2 - Start completed.
2023-05-21 16:55:33,746 INFO metastore.ObjectStore: Setting MetaStore object pin classes with hive.metastore.cache.pinobjtypes="Table,StorageDescriptor,SerDeInfo,Partition,Database,Type,FieldSchema,Order"
2023-05-21 16:55:33,870 INFO metastore.MetaStoreDirectSql: Using direct SQL, underlying DB is DERBY
2023-05-21 16:55:33,871 INFO metastore.ObjectStore: Initialized ObjectStore
```



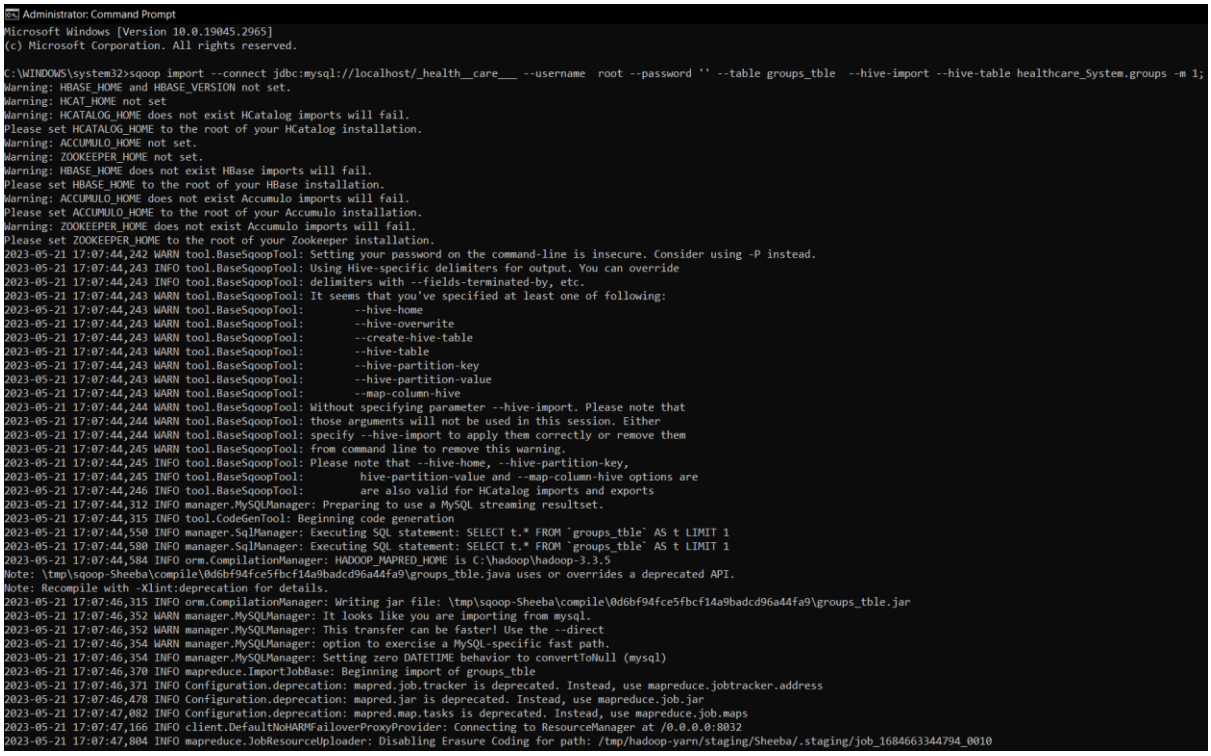
```
Administrator: Command Prompt - hive  --service hiveserver2 start
2023-05-21 17:19:38,920 INFO metastore.HiveMetaStore: 2: get_config_value: name=metastore.batch.retrieve.max defaultValue=50
2023-05-21 17:19:38,920 INFO HiveMetaStore.audit: ugi=Sheeba    ip=unknown-ip-addr      cmd=get_config_value: name=metastore.batch.retrieve.max defaultValue=50
2023-05-21 17:20:38,917 INFO metastore.HiveMetaStore: 2: get_config_value: name=metastore.batch.retrieve.max defaultValue=50
2023-05-21 17:20:38,917 INFO HiveMetaStore.audit: ugi=Sheeba    ip=unknown-ip-addr      cmd=get_config_value: name=metastore.batch.retrieve.max defaultValue=50
2023-05-21 17:21:38,908 INFO metastore.HiveMetaStore: 2: get_config_value: name=metastore.batch.retrieve.max defaultValue=50
2023-05-21 17:21:38,909 INFO HiveMetaStore.audit: ugi=Sheeba    ip=unknown-ip-addr      cmd=get_config_value: name=metastore.batch.retrieve.max defaultValue=50
2023-05-21 17:22:38,923 INFO metastore.HiveMetaStore: 2: get_config_value: name=metastore.batch.retrieve.max defaultValue=50
2023-05-21 17:22:38,923 INFO HiveMetaStore.audit: ugi=Sheeba    ip=unknown-ip-addr      cmd=get_config_value: name=metastore.batch.retrieve.max defaultValue=50
2023-05-21 17:23:38,925 INFO metastore.HiveMetaStore: 2: get_config_value: name=metastore.batch.retrieve.max defaultValue=50
2023-05-21 17:23:38,925 INFO HiveMetaStore.audit: ugi=Sheeba    ip=unknown-ip-addr      cmd=get_config_value: name=metastore.batch.retrieve.max defaultValue=50
2023-05-21 17:24:38,912 INFO metastore.HiveMetaStore: 2: get_config_value: name=metastore.batch.retrieve.max defaultValue=50
2023-05-21 17:24:38,912 INFO HiveMetaStore.audit: ugi=Sheeba    ip=unknown-ip-addr      cmd=get_config_value: name=metastore.batch.retrieve.max defaultValue=50
2023-05-21 17:25:38,921 INFO metastore.HiveMetaStore: 2: get_config_value: name=metastore.batch.retrieve.max defaultValue=50
2023-05-21 17:25:38,921 INFO HiveMetaStore.audit: ugi=Sheeba    ip=unknown-ip-addr      cmd=get_config_value: name=metastore.batch.retrieve.max defaultValue=50
2023-05-21 17:26:38,921 INFO metastore.HiveMetaStore: 2: get_config_value: name=metastore.batch.retrieve.max defaultValue=50
2023-05-21 17:26:38,921 INFO HiveMetaStore.audit: ugi=Sheeba    ip=unknown-ip-addr      cmd=get_config_value: name=metastore.batch.retrieve.max defaultValue=50
2023-05-21 17:27:38,917 INFO metastore.HiveMetaStore: 2: get_config_value: name=metastore.batch.retrieve.max defaultValue=50
2023-05-21 17:27:38,918 INFO HiveMetaStore.audit: ugi=Sheeba    ip=unknown-ip-addr      cmd=get_config_value: name=metastore.batch.retrieve.max defaultValue=50
2023-05-21 17:28:38,912 INFO metastore.HiveMetaStore: 2: get_config_value: name=metastore.batch.retrieve.max defaultValue=50
2023-05-21 17:28:38,913 INFO HiveMetaStore.audit: ugi=Sheeba    ip=unknown-ip-addr      cmd=get_config_value: name=metastore.batch.retrieve.max defaultValue=50
2023-05-21 17:29:38,915 INFO metastore.HiveMetaStore: 2: get_config_value: name=metastore.batch.retrieve.max defaultValue=50
2023-05-21 17:29:38,916 INFO HiveMetaStore.audit: ugi=Sheeba    ip=unknown-ip-addr      cmd=get_config_value: name=metastore.batch.retrieve.max defaultValue=50
2023-05-21 17:30:38,913 INFO metastore.HiveMetaStore: 2: get_config_value: name=metastore.batch.retrieve.max defaultValue=50
2023-05-21 17:30:38,913 INFO HiveMetaStore.audit: ugi=Sheeba    ip=unknown-ip-addr      cmd=get_config_value: name=metastore.batch.retrieve.max defaultValue=50
2023-05-21 17:31:38,916 INFO metastore.HiveMetaStore: 2: get_config_value: name=metastore.batch.retrieve.max defaultValue=50
2023-05-21 17:31:38,917 INFO HiveMetaStore.audit: ugi=Sheeba    ip=unknown-ip-addr      cmd=get_config_value: name=metastore.batch.retrieve.max defaultValue=50
2023-05-21 17:32:38,933 INFO metastore.HiveMetaStore: 2: get_config_value: name=metastore.batch.retrieve.max defaultValue=50
2023-05-21 17:32:38,933 INFO HiveMetaStore.audit: ugi=Sheeba    ip=unknown-ip-addr      cmd=get_config_value: name=metastore.batch.retrieve.max defaultValue=50
2023-05-21 17:33:38,917 INFO metastore.HiveMetaStore: 2: get_config_value: name=metastore.batch.retrieve.max defaultValue=50
2023-05-21 17:33:38,917 INFO HiveMetaStore.audit: ugi=Sheeba    ip=unknown-ip-addr      cmd=get_config_value: name=metastore.batch.retrieve.max defaultValue=50
2023-05-21 17:34:38,911 INFO metastore.HiveMetaStore: 2: get_config_value: name=metastore.batch.retrieve.max defaultValue=50
2023-05-21 17:34:38,911 INFO HiveMetaStore.audit: ugi=Sheeba    ip=unknown-ip-addr      cmd=get_config_value: name=metastore.batch.retrieve.max defaultValue=50
2023-05-21 17:35:38,911 INFO metastore.HiveMetaStore: 2: get_config_value: name=metastore.batch.retrieve.max defaultValue=50
2023-05-21 17:35:38,911 INFO HiveMetaStore.audit: ugi=Sheeba    ip=unknown-ip-addr      cmd=get_config_value: name=metastore.batch.retrieve.max defaultValue=50
```

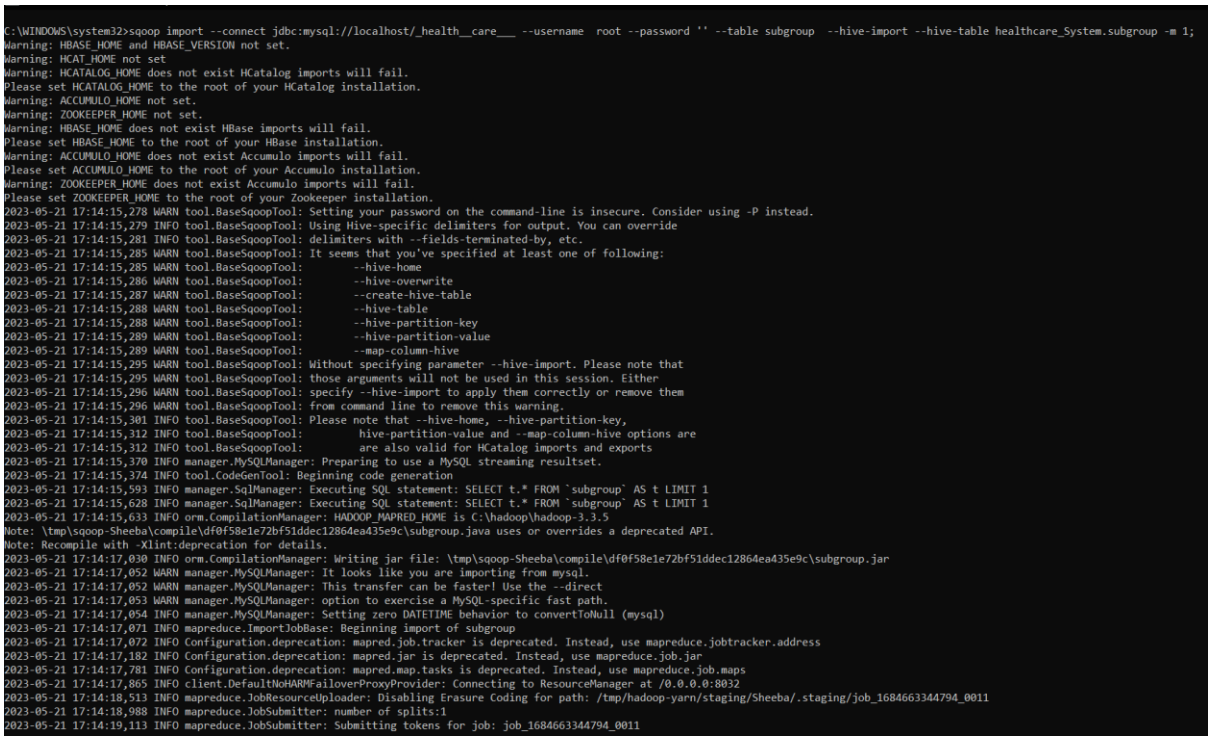
Database healthcare_System is created in hive:

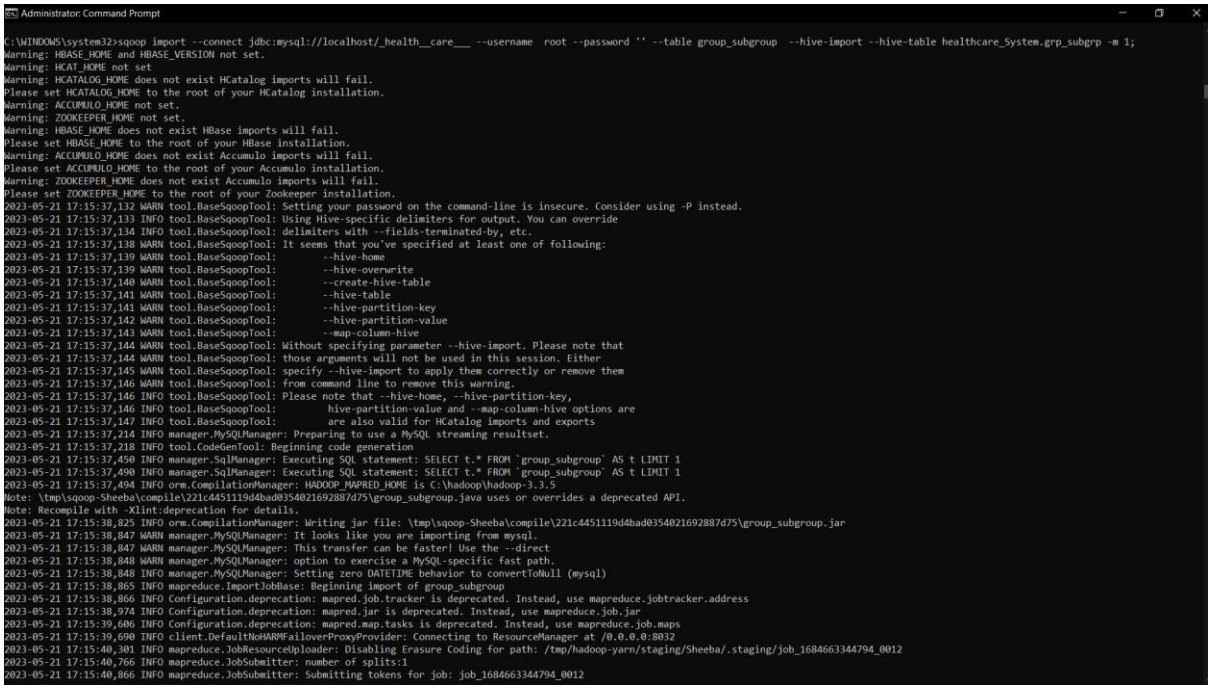


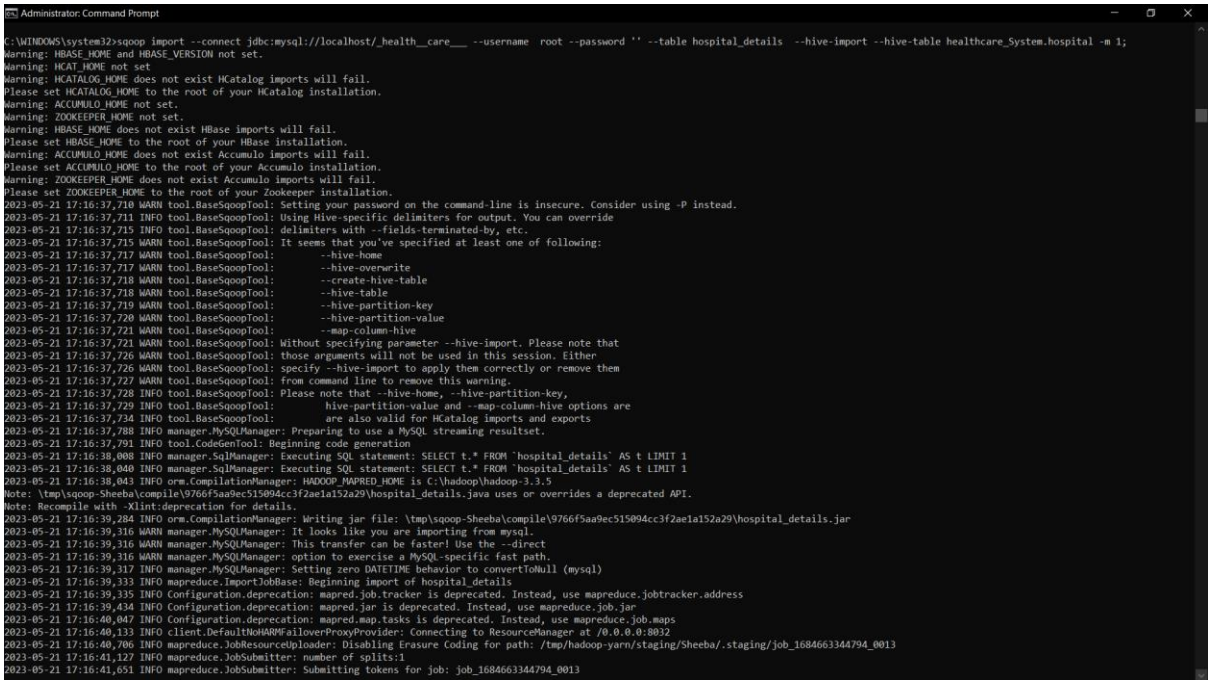


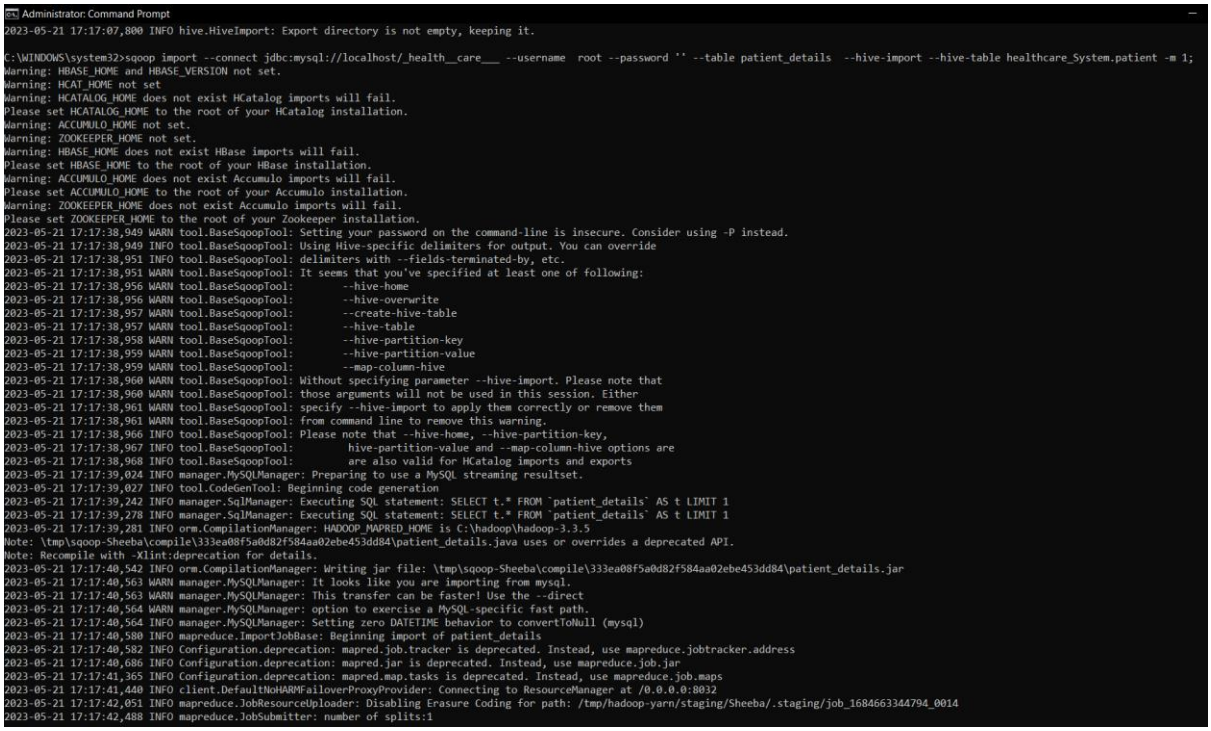
Importing the mysql tables using sqoop into hive tables:

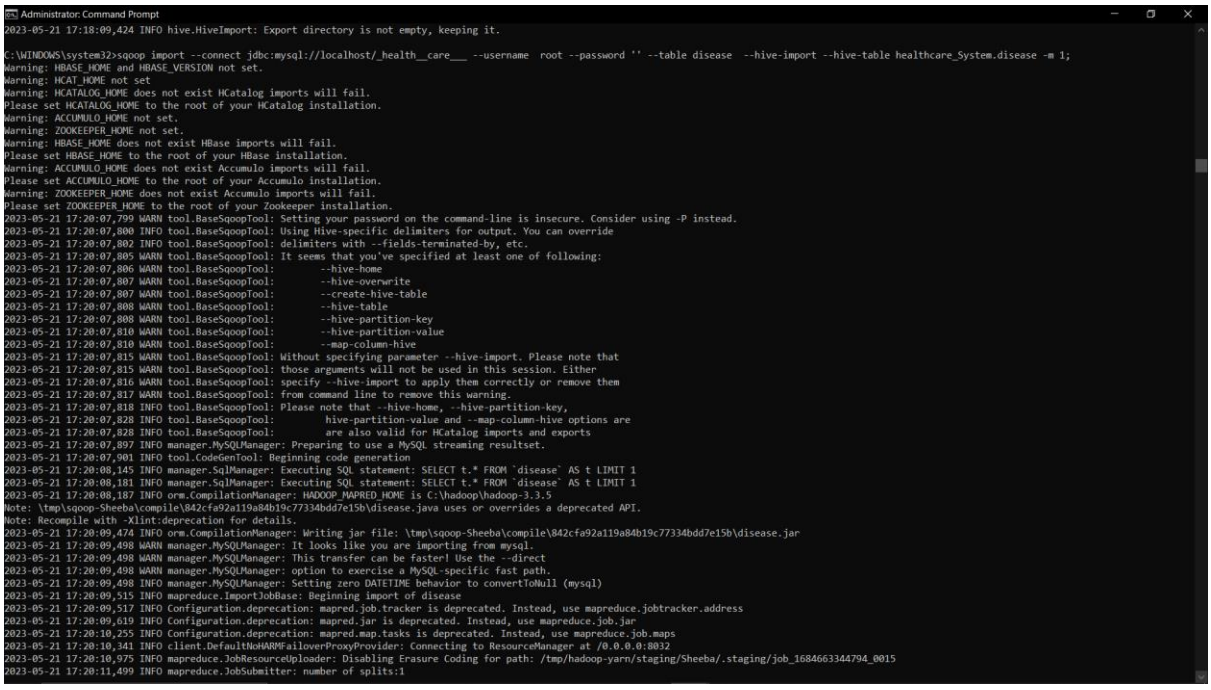


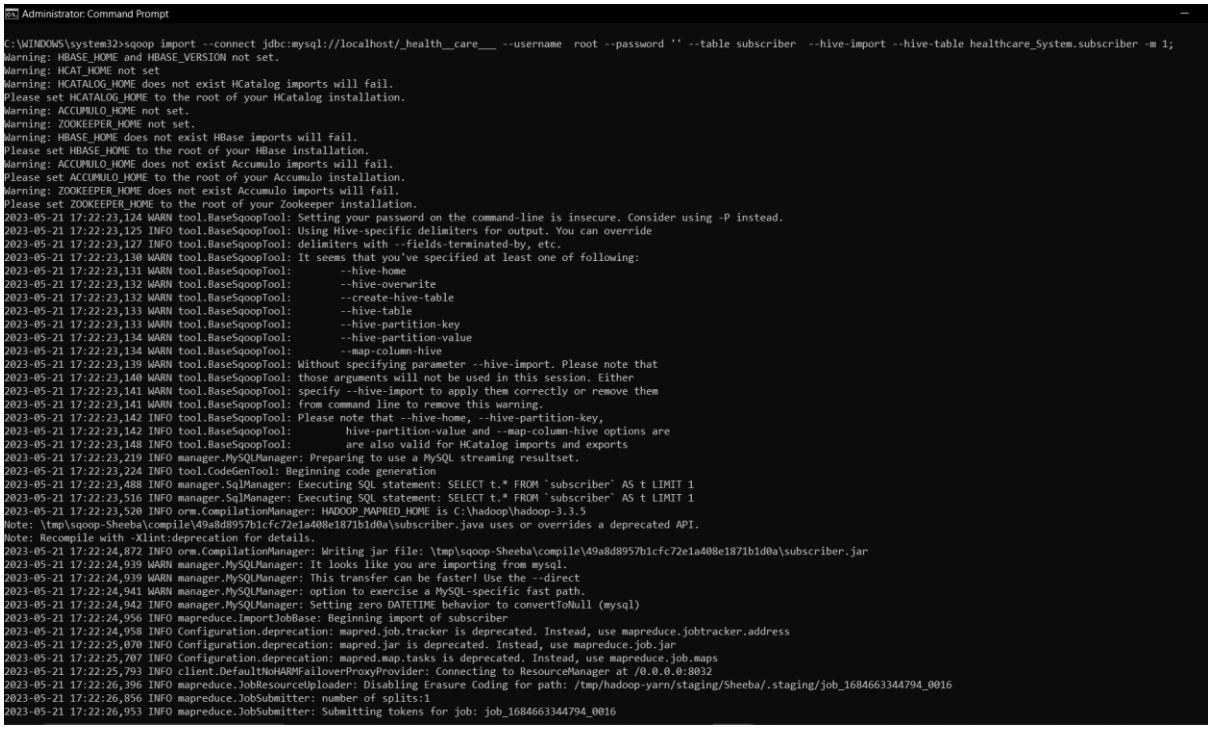


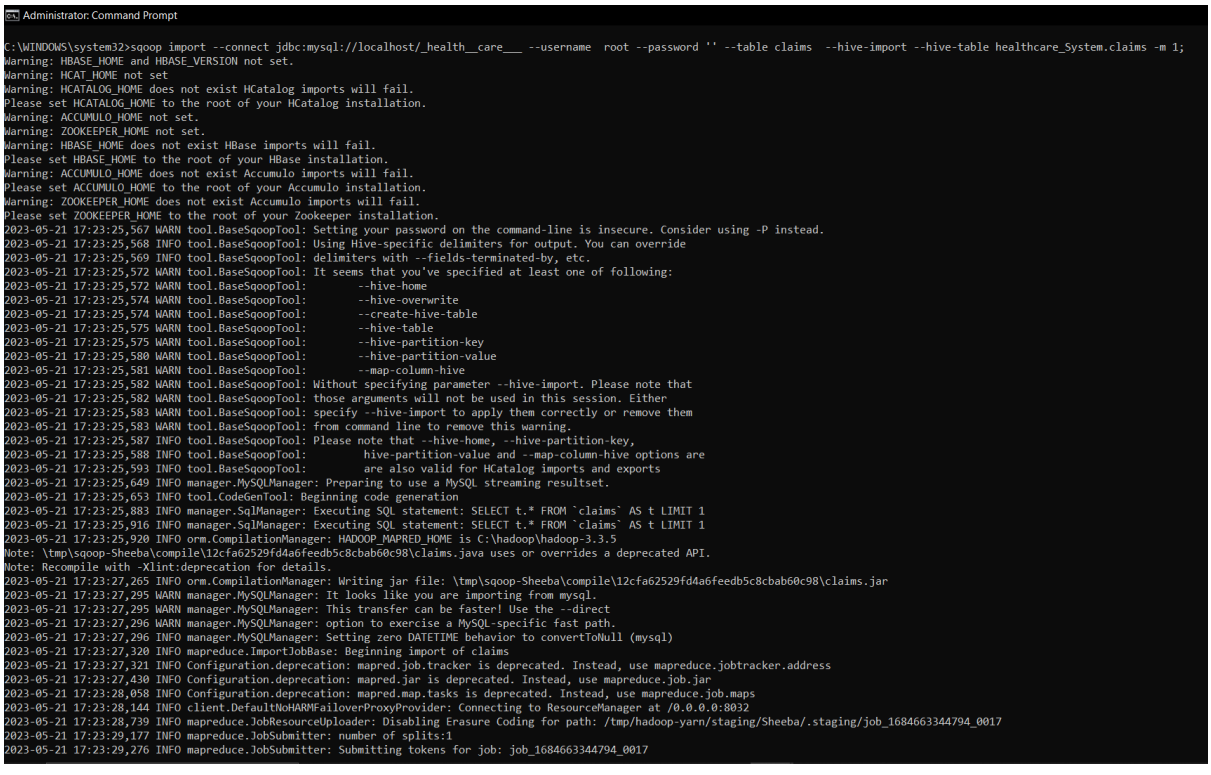

Administrator: Command Prompt
C:\WINDOWS\system32>sqoop import --connect jdbc:mysql://localhost/_health__care___ --username root --password '' --table group_subgroup --hive-import --hive-table healthcare_System.grp_subgrp -m 1;
Warning: HBASE_HOME and HBASE_VERSION not set.
Warning: HCAT_HOME not set
Warning: HCATALOG_HOME does not exist HCatalog imports will fail.
Please set HCATALOG_HOME to the root of your HCatalog installation.
Warning: ACCUMULO_HOME not set.
Warning: ZOOKEEPER_HOME not set.
Warning: HBASE_HOME does not exist HBase imports will fail.
Please set HBASE_HOME to the root of your HBase installation.
Warning: ACCUMULO_HOME does not exist Accumulo imports will fail.
Please set ACCUMULO_HOME to the root of your Accumulo installation.
Warning: ZOOKEEPER_HOME does not exist Accumulo imports will fail.
Please set ZOOKEEPER_HOME to the root of your Zookeeper installation.
2023-05-21 17:15:37,132 WARN tool.BaseSqoopTool: Setting your password on the command-line is insecure. Consider using -P instead.
2023-05-21 17:15:37,133 INFO tool.BaseSqoopTool: Using Hive-specific delimiters for output. You can override
2023-05-21 17:15:37,134 INFO tool.BaseSqoopTool: delimiters with --fields-terminated-by, etc.
2023-05-21 17:15:37,138 WARN tool.BaseSqoopTool: It seems that you've specified at least one of following:
2023-05-21 17:15:37,139 WARN tool.BaseSqoopTool: --hive-home
2023-05-21 17:15:37,139 WARN tool.BaseSqoopTool: --hive-overwrite
2023-05-21 17:15:37,140 WARN tool.BaseSqoopTool: --create-hive-table
2023-05-21 17:15:37,141 WARN tool.BaseSqoopTool: --hive-table
2023-05-21 17:15:37,141 WARN tool.BaseSqoopTool: --hive-partition-key
2023-05-21 17:15:37,142 WARN tool.BaseSqoopTool: --hive-partition-value
2023-05-21 17:15:37,143 WARN tool.BaseSqoopTool: --map-column-hive
2023-05-21 17:15:37,144 WARN tool.BaseSqoopTool: Without specifying parameter --hive-import. Please note that
2023-05-21 17:15:37,144 WARN tool.BaseSqoopTool: those arguments will not be used in this session. Either
2023-05-21 17:15:37,145 WARN tool.BaseSqoopTool: specify --hive-import to apply them correctly or remove them
2023-05-21 17:15:37,146 WARN tool.BaseSqoopTool: from command line to remove this warning.
2023-05-21 17:15:37,146 INFO tool.BaseSqoopTool: Please note that --hive-home, --hive-partition-key,
2023-05-21 17:15:37,146 INFO tool.BaseSqoopTool: hive-partition-value and --map-column-hive options are
2023-05-21 17:15:37,147 INFO tool.BaseSqoopTool: are also valid for HCatalog imports and exports
2023-05-21 17:15:37,214 INFO manager.MySQLManager: Preparing to use a MySQL streaming resultset.
2023-05-21 17:15:37,218 INFO tool.CodeGenTool: Beginning code generation
2023-05-21 17:15:37,450 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM `group_subgroup` AS t LIMIT 1
2023-05-21 17:15:37,490 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM `group_subgroup` AS t LIMIT 1
2023-05-21 17:15:37,494 INFO orm.CompilationManager: HADOOP_MAPRED_HOME is C:\hadoop\hadoop-3.3.5
Note: \tmp\sqoop-Sheeba\compile\221c4451119d4bad03540216928870d75\group_subgroup.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
2023-05-21 17:15:38,825 INFO orm.CompilationManager: Writing jar file: \tmp\sqoop-Sheeba\compile\221c4451119d4bad03540216928870d75\group_subgroup.jar
2023-05-21 17:15:38,847 WARN manager.MySQLManager: It looks like you are importing from mysql.
2023-05-21 17:15:38,847 WARN manager.MySQLManager: This transfer can be faster! Use the --direct
2023-05-21 17:15:38,848 WARN manager.MySQLManager: option to exercise a MySQL-specific fast path.
2023-05-21 17:15:38,848 INFO manager.MySQLManager: Setting zero DATETIME behavior to convertToNull (mysql)
2023-05-21 17:15:38,865 INFO mapreduce.ImportJobBase: Beginning import of group_subgroup
2023-05-21 17:15:38,866 INFO Configuration.deprecation: mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
2023-05-21 17:15:38,974 INFO Configuration.deprecation: mapred.jar is deprecated. Instead, use mapreduce.job.jar
2023-05-21 17:15:39,606 INFO Configuration.deprecation: mapred.map.tasks is deprecated. Instead, use mapreduce.job.maps
2023-05-21 17:15:39,690 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at /0.0.0.0:8032
2023-05-21 17:15:40,301 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/Sheeba/.staging/job_1684663344794_0012
2023-05-21 17:15:40,766 INFO mapreduce.JobSubmitter: number of splits:1
2023-05-21 17:15:40,866 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1684663344794_0012


Administrator: Command Prompt
C:\WINDOWS\system32>sqoop import --connect jdbc:mysql://localhost/_health__care___ --username root --password '' --table hospital_details --hive-import --hive-table healthcare_System.hospital -m 1;
Warning: HBASE_HOME and HBASE_VERSION not set.
Warning: HCAT_HOME not set
Warning: HCATALOG_HOME does not exist HCatalog imports will fail.
Please set HCATALOG_HOME to the root of your HCatalog installation.
Warning: ACCUMULO_HOME not set.
Warning: ZOOKEEPER_HOME not set.
Warning: HBASE_HOME does not exist HBase imports will fail.
Please set HBASE_HOME to the root of your HBase installation.
Warning: ACCUMULO_HOME does not exist Accumulo imports will fail.
Please set ACCUMULO_HOME to the root of your Accumulo installation.
Warning: ZOOKEEPER_HOME does not exist Accumulo imports will fail.
Please set ZOOKEEPER_HOME to the root of your Zookeeper installation.
2023-05-21 17:16:37,710 WARN tool.BaseSqoopTool: Setting your password on the command-line is insecure. Consider using -P instead.
2023-05-21 17:16:37,711 INFO tool.BaseSqoopTool: Using Hive-specific delimiters for output. You can override
2023-05-21 17:16:37,715 INFO tool.BaseSqoopTool: delimiters with --fields-terminated-by, etc.
2023-05-21 17:16:37,715 WARN tool.BaseSqoopTool: It seems that you've specified at least one of following:
2023-05-21 17:16:37,717 WARN tool.BaseSqoopTool: --hive-home
2023-05-21 17:16:37,717 WARN tool.BaseSqoopTool: --hive-overwrite
2023-05-21 17:16:37,718 WARN tool.BaseSqoopTool: --create-hive-table
2023-05-21 17:16:37,718 WARN tool.BaseSqoopTool: --hive-table
2023-05-21 17:16:37,719 WARN tool.BaseSqoopTool: --hive-partition-key
2023-05-21 17:16:37,720 WARN tool.BaseSqoopTool: --hive-partition-value
2023-05-21 17:16:37,721 WARN tool.BaseSqoopTool: --map-column-hive
2023-05-21 17:16:37,721 WARN tool.BaseSqoopTool: Without specifying parameter --hive-import. Please note that
2023-05-21 17:16:37,726 WARN tool.BaseSqoopTool: those arguments will not be used in this session. Either
2023-05-21 17:16:37,726 WARN tool.BaseSqoopTool: specify --hive-import to apply them correctly or remove them
2023-05-21 17:16:37,727 WARN tool.BaseSqoopTool: from command line to remove this warning.
2023-05-21 17:16:37,728 INFO tool.BaseSqoopTool: Please note that --hive-home, --hive-partition-key,
2023-05-21 17:16:37,729 INFO tool.BaseSqoopTool: hive-partition-value and --map-column-hive options are
2023-05-21 17:16:37,734 INFO tool.BaseSqoopTool: are also valid for HCatalog imports and exports
2023-05-21 17:16:37,788 INFO manager.MySQLManager: Preparing to use a MySQL streaming resultset.
2023-05-21 17:16:37,791 INFO tool.CodeGenTool: Beginning code generation
2023-05-21 17:16:38,008 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM `hospital_details` AS t LIMIT 1
2023-05-21 17:16:38,040 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM `hospital_details` AS t LIMIT 1
2023-05-21 17:16:38,043 INFO orm.CompilationManager: HADOOP_MAPRED_HOME is C:\hadoop\hadoop-3.3.5
Note: \tmp\sqoop-Sheeba\compile\9766f5aa9ec515094cc3f2ae1a152a29\hospital_details.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
2023-05-21 17:16:39,284 INFO orm.CompilationManager: Writing jar file: \tmp\sqoop-Sheeba\compile\9766f5aa9ec515094cc3f2ae1a152a29\hospital_details.jar
2023-05-21 17:16:39,316 WARN manager.MySQLManager: It looks like you are importing from mysql.
2023-05-21 17:16:39,316 WARN manager.MySQLManager: This transfer can be faster! Use the --direct
2023-05-21 17:16:39,316 WARN manager.MySQLManager: option to exercise a MySQL-specific fast path.
2023-05-21 17:16:39,317 INFO manager.MySQLManager: Setting zero DATETIME behavior to convertToNull (mysql)
2023-05-21 17:16:39,333 INFO mapreduce.ImportJobBase: Beginning import of hospital_details
2023-05-21 17:16:39,335 INFO Configuration.deprecation: mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
2023-05-21 17:16:39,434 INFO Configuration.deprecation: mapred.jar is deprecated. Instead, use mapreduce.job.jar
2023-05-21 17:16:40,047 INFO Configuration.deprecation: mapred.map.tasks is deprecated. Instead, use mapreduce.job.maps
2023-05-21 17:16:40,133 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at /0.0.0.0:8032
2023-05-21 17:16:40,706 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/Sheeba/.staging/job_1684663344794_0013
2023-05-21 17:16:41,127 INFO mapreduce.JobSubmitter: number of splits:1
2023-05-21 17:16:41,651 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1684663344794_0013

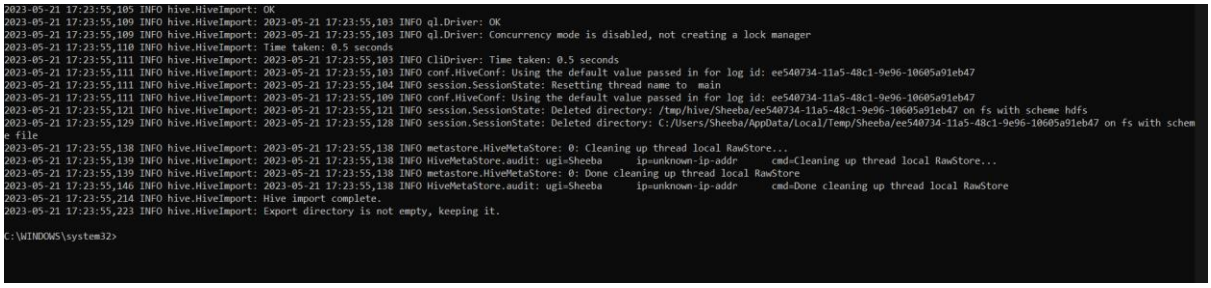

Administrator: Command Prompt
2023-05-21 17:17:07,800 INFO hive.HiveImport: Export directory is not empty, keeping it.
C:\WINDOWS\system32>sqoop import --connect jdbc:mysql://localhost/_health__care___ --username root --password '' --table patient_details --hive-import --hive-table healthcare_System.patient -m 1;
Warning: HBASE_HOME and HBASE_VERSION not set.
Warning: HCAT_HOME not set
Warning: HCATALOG_HOME does not exist HCatalog imports will fail.
Please set HCATALOG_HOME to the root of your HCatalog installation.
Warning: ACCUMULO_HOME not set.
Warning: ZOOKEEPER_HOME not set.
Warning: HBASE_HOME does not exist HBase imports will fail.
Please set HBASE_HOME to the root of your HBase installation.
Warning: ACCUMULO_HOME does not exist Accumulo imports will fail.
Please set ACCUMULO_HOME to the root of your Accumulo installation.
Warning: ZOOKEEPER_HOME does not exist Accumulo imports will fail.
Please set ZOOKEEPER_HOME to the root of your Zookeeper installation.
2023-05-21 17:17:38,949 WARN tool.BaseSqoopTool: Setting your password on the command-line is insecure. Consider using -P instead.
2023-05-21 17:17:38,949 INFO tool.BaseSqoopTool: Using Hive-specific delimiters for output. You can override
2023-05-21 17:17:38,951 INFO tool.BaseSqoopTool: delimiters with --fields-terminated-by, etc.
2023-05-21 17:17:38,951 WARN tool.BaseSqoopTool: It seems that you've specified at least one of following:
2023-05-21 17:17:38,956 WARN tool.BaseSqoopTool: --hive-home
2023-05-21 17:17:38,956 WARN tool.BaseSqoopTool: --hive-overwrite
2023-05-21 17:17:38,957 WARN tool.BaseSqoopTool: --create-hive-table
2023-05-21 17:17:38,957 WARN tool.BaseSqoopTool: --hive-table
2023-05-21 17:17:38,958 WARN tool.BaseSqoopTool: --hive-partition-key
2023-05-21 17:17:38,959 WARN tool.BaseSqoopTool: --hive-partition-value
2023-05-21 17:17:38,959 WARN tool.BaseSqoopTool: --map-column-hive
2023-05-21 17:17:38,960 WARN tool.BaseSqoopTool: Without specifying parameter --hive-import. Please note that
2023-05-21 17:17:38,960 WARN tool.BaseSqoopTool: those arguments will not be used in this session. Either
2023-05-21 17:17:38,961 WARN tool.BaseSqoopTool: specify --hive-import to apply them correctly or remove them
2023-05-21 17:17:38,961 WARN tool.BaseSqoopTool: from command line to remove this warning.
2023-05-21 17:17:38,966 INFO tool.BaseSqoopTool: Please note that --hive-home, --hive-partition-key,
2023-05-21 17:17:38,967 INFO tool.BaseSqoopTool: hive-partition-value and --map-column-hive options are
2023-05-21 17:17:38,968 INFO tool.BaseSqoopTool: are also valid for HCatalog imports and exports
2023-05-21 17:17:39,024 INFO manager.MySQLManager: Preparing to use a MySQL streaming resultset.
2023-05-21 17:17:39,027 INFO tool.CodeGenTool: Beginning code generation
2023-05-21 17:17:39,242 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM `patient_details` AS t LIMIT 1
2023-05-21 17:17:39,278 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM `patient_details` AS t LIMIT 1
2023-05-21 17:17:39,281 INFO orm.CompilationManager: HADOOP_MAPRED_HOME is C:\hadoop\hadoop-3.3.5
Note: \tmp\sqoop-Sheeba\compile\333ea08f5a0d82f584aa02ebe453dd84\patient_details.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
2023-05-21 17:17:40,542 INFO orm.CompilationManager: Writing jar file: \tmp\sqoop-Sheeba\compile\333ea08f5a0d82f584aa02ebe453dd84\patient_details.jar
2023-05-21 17:17:40,563 WARN manager.MySQLManager: It looks like you are importing from mysql.
2023-05-21 17:17:40,563 WARN manager.MySQLManager: This transfer can be faster! Use the --direct
2023-05-21 17:17:40,564 WARN manager.MySQLManager: option to exercise a MySQL-specific fast path.
2023-05-21 17:17:40,564 INFO manager.MySQLManager: Setting zero DATETIME behavior to convertToNull (mysql)
2023-05-21 17:17:40,580 INFO mapreduce.ImportJobBase: Beginning import of patient_details
2023-05-21 17:17:40,582 INFO Configuration.deprecation: mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
2023-05-21 17:17:40,686 INFO Configuration.deprecation: mapred.jar is deprecated. Instead, use mapreduce.job.jar
2023-05-21 17:17:41,365 INFO Configuration.deprecation: mapred.map.tasks is deprecated. Instead, use mapreduce.job.maps
2023-05-21 17:17:41,440 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at /0.0.0.0:8032
2023-05-21 17:17:42,051 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/Sheeba/.staging/job_1684663344794_0014
2023-05-21 17:17:42,488 INFO mapreduce.JobSubmitter: number of splits:1


Administrator: Command Prompt
2023-05-21 17:18:09,424 INFO hive.HiveImport: Export directory is not empty, keeping it.
C:\WINDOWS\system32>sqoop import --connect jdbc:mysql://localhost/_health__care___ --username root --password '' --table disease --hive-import --hive-table healthcare_System.disease -m 1;
Warning: HBASE_HOME and HBASE_VERSION not set.
Warning: HCAT_HOME not set
Warning: HCATALOG_HOME does not exist HCatalog imports will fail.
Please set HCATALOG_HOME to the root of your HCatalog installation.
Warning: ACCUMULO_HOME not set.
Warning: ZOOKEEPER_HOME not set.
Warning: HBASE_HOME does not exist HBase imports will fail.
Please set HBASE_HOME to the root of your HBase installation.
Warning: ACCUMULO_HOME does not exist Accumulo imports will fail.
Please set ACCUMULO_HOME to the root of your Accumulo installation.
Warning: ZOOKEEPER_HOME does not exist Accumulo imports will fail.
Please set ZOOKEEPER_HOME to the root of your Zookeeper installation.
2023-05-21 17:20:07,799 WARN tool.BaseSqoopTool: Setting your password on the command-line is insecure. Consider using -P instead.
2023-05-21 17:20:07,800 INFO tool.BaseSqoopTool: Using Hive-specific delimiters for output. You can override
2023-05-21 17:20:07,802 INFO tool.BaseSqoopTool: delimiters with --fields-terminated-by, etc.
2023-05-21 17:20:07,805 WARN tool.BaseSqoopTool: It seems that you've specified at least one of following:
2023-05-21 17:20:07,806 WARN tool.BaseSqoopTool: --hive-home
2023-05-21 17:20:07,807 WARN tool.BaseSqoopTool: --hive-overwrite
2023-05-21 17:20:07,807 WARN tool.BaseSqoopTool: --create-hive-table
2023-05-21 17:20:07,808 WARN tool.BaseSqoopTool: --hive-table
2023-05-21 17:20:07,808 WARN tool.BaseSqoopTool: --hive-partition-key
2023-05-21 17:20:07,810 WARN tool.BaseSqoopTool: --hive-partition-value
2023-05-21 17:20:07,810 WARN tool.BaseSqoopTool: --map-column-hive
2023-05-21 17:20:07,815 WARN tool.BaseSqoopTool: Without specifying parameter --hive-import. Please note that
2023-05-21 17:20:07,815 WARN tool.BaseSqoopTool: those arguments will not be used in this session. Either
2023-05-21 17:20:07,816 WARN tool.BaseSqoopTool: specify --hive-import to apply them correctly or remove them
2023-05-21 17:20:07,817 WARN tool.BaseSqoopTool: from command line to remove this warning.
2023-05-21 17:20:07,818 INFO tool.BaseSqoopTool: Please note that --hive-home, --hive-partition-key,
2023-05-21 17:20:07,828 INFO tool.BaseSqoopTool: hive-partition-value and --map-column-hive options are
2023-05-21 17:20:07,828 INFO tool.BaseSqoopTool: are also valid for HCatalog imports and exports
2023-05-21 17:20:07,897 INFO manager.MySQLManager: Preparing to use a MySQL streaming resultset.
2023-05-21 17:20:07,901 INFO tool.CodeGenTool: Beginning code generation
2023-05-21 17:20:08,145 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM `disease` AS t LIMIT 1
2023-05-21 17:20:08,181 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM `disease` AS t LIMIT 1
2023-05-21 17:20:08,187 INFO orm.CompilationManager: HADOOP_MAPRED_HOME is C:\hadoop\hadoop-3.3.5
Note: \tmp\sqoop-Sheeba\compile\842cfa92a119a84b19c77334bdd7e15b\disease.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
2023-05-21 17:20:09,474 INFO orm.CompilationManager: Writing jar file: \tmp\sqoop-Sheeba\compile\842cfa92a119a84b19c77334bdd7e15b\disease.jar
2023-05-21 17:20:09,498 WARN manager.MySQLManager: It looks like you are importing from mysql.
2023-05-21 17:20:09,498 WARN manager.MySQLManager: This transfer can be faster! Use the --direct
2023-05-21 17:20:09,498 WARN manager.MySQLManager: option to exercise a MySQL-specific fast path.
2023-05-21 17:20:09,498 INFO manager.MySQLManager: Setting zero DATETIME behavior to convertToNull (mysql)
2023-05-21 17:20:09,515 INFO mapreduce.ImportJobBase: Beginning import of disease
2023-05-21 17:20:09,517 INFO Configuration.deprecation: mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
2023-05-21 17:20:09,619 INFO Configuration.deprecation: mapred.jar is deprecated. Instead, use mapreduce.job.jar
2023-05-21 17:20:10,255 INFO Configuration.deprecation: mapred.map.tasks is deprecated. Instead, use mapreduce.job.maps
2023-05-21 17:20:10,341 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at /0.0.0.0:8032
2023-05-21 17:20:10,975 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/Sheeba/.staging/job_1684663344794_0015
2023-05-21 17:20:11,499 INFO mapreduce.JobSubmitter: number of splits:1


Administrator: Command Prompt
C:\WINDOWS\system32>sqoop import --connect jdbc:mysql://localhost/_health__care___ --username root --password '' --table subscriber --hive-import --hive-table healthcare_System.subscriber -m 1;
Warning: HBASE_HOME and HBASE_VERSION not set.
Warning: HCAT_HOME not set
Warning: HCATALOG_HOME does not exist HCatalog imports will fail.
Please set HCATALOG_HOME to the root of your HCatalog installation.
Warning: ACCUMULO_HOME not set.
Warning: ZOOKEEPER_HOME not set.
Warning: HBASE_HOME does not exist HBase imports will fail.
Please set HBASE_HOME to the root of your HBase installation.
Warning: ACCUMULO_HOME does not exist Accumulo imports will fail.
Please set ACCUMULO_HOME to the root of your Accumulo installation.
Warning: ZOOKEEPER_HOME does not exist Accumulo imports will fail.
Please set ZOOKEEPER_HOME to the root of your Zookeeper installation.
2023-05-21 17:22:23,124 WARN tool.BaseSqoopTool: Setting your password on the command-line is insecure. Consider using -P instead.
2023-05-21 17:22:23,125 INFO tool.BaseSqoopTool: Using Hive-specific delimiters for output. You can override
2023-05-21 17:22:23,127 INFO tool.BaseSqoopTool: delimiters with --fields-terminated-by, etc.
2023-05-21 17:22:23,130 WARN tool.BaseSqoopTool: It seems that you've specified at least one of following:
2023-05-21 17:22:23,131 WARN tool.BaseSqoopTool: --hive-home
2023-05-21 17:22:23,132 WARN tool.BaseSqoopTool: --hive-overwrite
2023-05-21 17:22:23,132 WARN tool.BaseSqoopTool: --create-hive-table
2023-05-21 17:22:23,133 WARN tool.BaseSqoopTool: --hive-table
2023-05-21 17:22:23,133 WARN tool.BaseSqoopTool: --hive-partition-key
2023-05-21 17:22:23,134 WARN tool.BaseSqoopTool: --hive-partition-value
2023-05-21 17:22:23,134 WARN tool.BaseSqoopTool: --map-column-hive
2023-05-21 17:22:23,139 WARN tool.BaseSqoopTool: Without specifying parameter --hive-import. Please note that
2023-05-21 17:22:23,140 WARN tool.BaseSqoopTool: those arguments will not be used in this session. Either
2023-05-21 17:22:23,141 WARN tool.BaseSqoopTool: specify --hive-import to apply them correctly or remove them
2023-05-21 17:22:23,141 WARN tool.BaseSqoopTool: from command line to remove this warning.
2023-05-21 17:22:23,142 INFO tool.BaseSqoopTool: Please note that --hive-home, --hive-partition-key,
2023-05-21 17:22:23,142 INFO tool.BaseSqoopTool: hive-partition-value and --map-column-hive options are
2023-05-21 17:22:23,148 INFO tool.BaseSqoopTool: are also valid for HCatalog imports and exports
2023-05-21 17:22:23,219 INFO manager.MySQLManager: Preparing to use a MySQL streaming resultset.
2023-05-21 17:22:23,224 INFO tool.CodeGenTool: Beginning code generation
2023-05-21 17:22:23,488 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM `subscriber` AS t LIMIT 1
2023-05-21 17:22:23,516 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM `subscriber` AS t LIMIT 1
2023-05-21 17:22:23,520 INFO orm.CompilationManager: HADOOP_MAPRED_HOME is C:\hadoop\hadoop-3.3.5
Note: \tmp\sqoop-Sheeba\compile\49a8d8957b1cfc72e1a408e1871b1d0a\subscriber.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
2023-05-21 17:22:24,872 INFO orm.CompilationManager: Writing jar file: \tmp\sqoop-Sheeba\compile\49a8d8957b1cfc72e1a408e1871b1d0a\subscriber.jar
2023-05-21 17:22:24,939 WARN manager.MySQLManager: It looks like you are importing from mysql.
2023-05-21 17:22:24,939 WARN manager.MySQLManager: This transfer can be faster! Use the --direct
2023-05-21 17:22:24,941 WARN manager.MySQLManager: option to exercise a MySQL-specific fast path.
2023-05-21 17:22:24,942 INFO manager.MySQLManager: Setting zero DATETIME behavior to convertToNull (mysql)
2023-05-21 17:22:24,956 INFO mapreduce.ImportJobBase: Beginning import of subscriber
2023-05-21 17:22:24,958 INFO Configuration.deprecation: mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
2023-05-21 17:22:25,070 INFO Configuration.deprecation: mapred.jar is deprecated. Instead, use mapreduce.job.jar
2023-05-21 17:22:25,707 INFO Configuration.deprecation: mapred.map.tasks is deprecated. Instead, use mapreduce.job.maps
2023-05-21 17:22:25,793 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at /0.0.0.0:8032
2023-05-21 17:22:26,396 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/Sheeba/.staging/job_1684663344794_0016
2023-05-21 17:22:26,856 INFO mapreduce.JobSubmitter: number of splits:1
2023-05-21 17:22:26,953 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1684663344794_0016

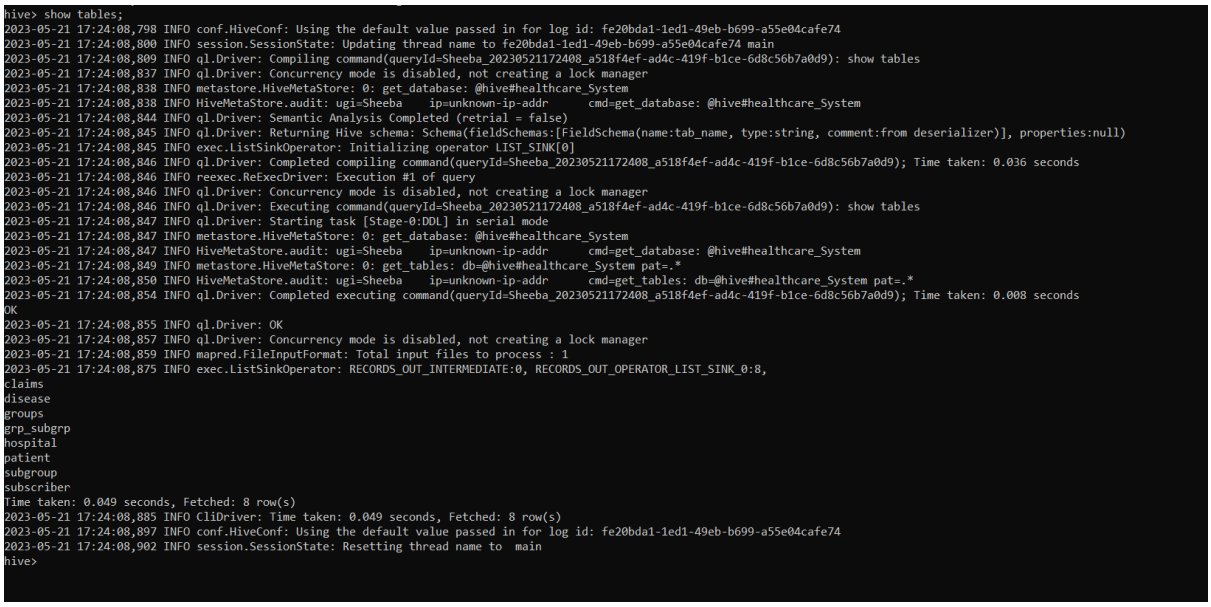

Administrator: Command Prompt
C:\WINDOWS\system32>sqoop import --connect jdbc:mysql://localhost/_health__care___ --username root --password '' --table claims --hive-import --hive-table healthcare_System.claims -m 1;
Warning: HBASE_HOME and HBASE_VERSION not set.
Warning: HCAT_HOME not set
Warning: HCATALOG_HOME does not exist HCatalog imports will fail.
Please set HCATALOG_HOME to the root of your HCatalog installation.
Warning: ACCUMULO_HOME not set.
Warning: ZOOKEEPER_HOME not set.
Warning: HBASE_HOME does not exist HBase imports will fail.
Please set HBASE_HOME to the root of your HBase installation.
Warning: ACCUMULO_HOME does not exist Accumulo imports will fail.
Please set ACCUMULO_HOME to the root of your Accumulo installation.
Warning: ZOOKEEPER_HOME does not exist Accumulo imports will fail.
Please set ZOOKEEPER_HOME to the root of your Zookeeper installation.
2023-05-21 17:23:25,567 WARN tool.BaseSqoopTool: Setting your password on the command-line is insecure. Consider using -P instead.
2023-05-21 17:23:25,568 INFO tool.BaseSqoopTool: Using Hive-specific delimiters for output. You can override
2023-05-21 17:23:25,569 INFO tool.BaseSqoopTool: delimiters with --fields-terminated-by, etc.
2023-05-21 17:23:25,572 WARN tool.BaseSqoopTool: It seems that you've specified at least one of following:
2023-05-21 17:23:25,572 WARN tool.BaseSqoopTool: --hive-home
2023-05-21 17:23:25,574 WARN tool.BaseSqoopTool: --hive-overwrite
2023-05-21 17:23:25,574 WARN tool.BaseSqoopTool: --create-hive-table
2023-05-21 17:23:25,575 WARN tool.BaseSqoopTool: --hive-table
2023-05-21 17:23:25,575 WARN tool.BaseSqoopTool: --hive-partition-key
2023-05-21 17:23:25,580 WARN tool.BaseSqoopTool: --hive-partition-value
2023-05-21 17:23:25,581 WARN tool.BaseSqoopTool: --map-column-hive
2023-05-21 17:23:25,582 WARN tool.BaseSqoopTool: Without specifying parameter --hive-import. Please note that
2023-05-21 17:23:25,582 WARN tool.BaseSqoopTool: those arguments will not be used in this session. Either
2023-05-21 17:23:25,583 WARN tool.BaseSqoopTool: specify --hive-import to apply them correctly or remove them
2023-05-21 17:23:25,583 WARN tool.BaseSqoopTool: from command line to remove this warning.
2023-05-21 17:23:25,587 INFO tool.BaseSqoopTool: Please note that --hive-home, --hive-partition-key,
2023-05-21 17:23:25,588 INFO tool.BaseSqoopTool: hive-partition-value and --map-column-hive options are
2023-05-21 17:23:25,593 INFO tool.BaseSqoopTool: are also valid for HCatalog imports and exports
2023-05-21 17:23:25,649 INFO manager.MySQLManager: Preparing to use a MySQL streaming resultset.
2023-05-21 17:23:25,653 INFO tool.CodeGenTool: Beginning code generation
2023-05-21 17:23:25,883 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM `claims` AS t LIMIT 1
2023-05-21 17:23:25,916 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM `claims` AS t LIMIT 1
2023-05-21 17:23:25,920 INFO orm.CompilationManager: HADOOP_MAPRED_HOME is C:\hadoop\hadoop-3.3.5
Note: \tmp\sqoop-Sheeba\compile\12cfa62529fd4a6feedb5c8cbab60c98\claims.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
2023-05-21 17:23:27,265 INFO orm.CompilationManager: Writing jar file: \tmp\sqoop-Sheeba\compile\12cfa62529fd4a6feedb5c8cbab60c98\claims.jar
2023-05-21 17:23:27,295 WARN manager.MySQLManager: It looks like you are importing from mysql.
2023-05-21 17:23:27,295 WARN manager.MySQLManager: This transfer can be faster! Use the --direct
2023-05-21 17:23:27,296 WARN manager.MySQLManager: option to exercise a MySQL-specific fast path.
2023-05-21 17:23:27,296 INFO manager.MySQLManager: Setting zero DATETIME behavior to convertToNull (mysql)
2023-05-21 17:23:27,320 INFO mapreduce.ImportJobBase: Beginning import of claims
2023-05-21 17:23:27,321 INFO Configuration.deprecation: mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
2023-05-21 17:23:27,430 INFO Configuration.deprecation: mapred.jar is deprecated. Instead, use mapreduce.job.jar
2023-05-21 17:23:28,058 INFO Configuration.deprecation: mapred.map.tasks is deprecated. Instead, use mapreduce.job.maps
2023-05-21 17:23:28,144 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at /0.0.0.0:8032
2023-05-21 17:23:28,739 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/Sheeba/.staging/job_1684663344794_0017
2023-05-21 17:23:29,177 INFO mapreduce.JobSubmitter: number of splits:1
2023-05-21 17:23:29,276 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1684663344794_0017

Hive import complete: Succesfully Imported tables from mysql to hive using sqoop

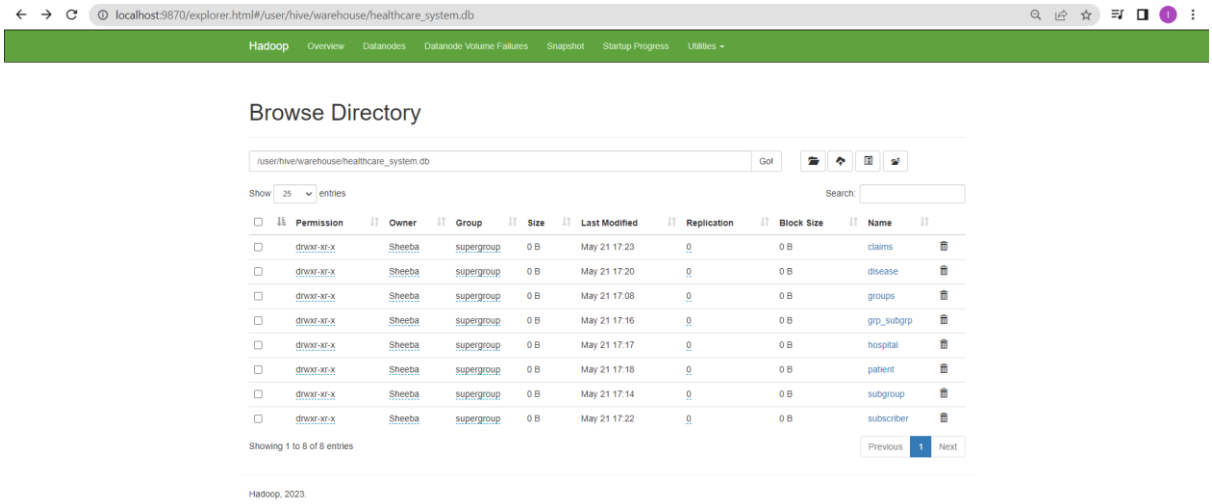


5.5 DATA STORAGE, PROCESSING AND ANALYTICS SERVICES:

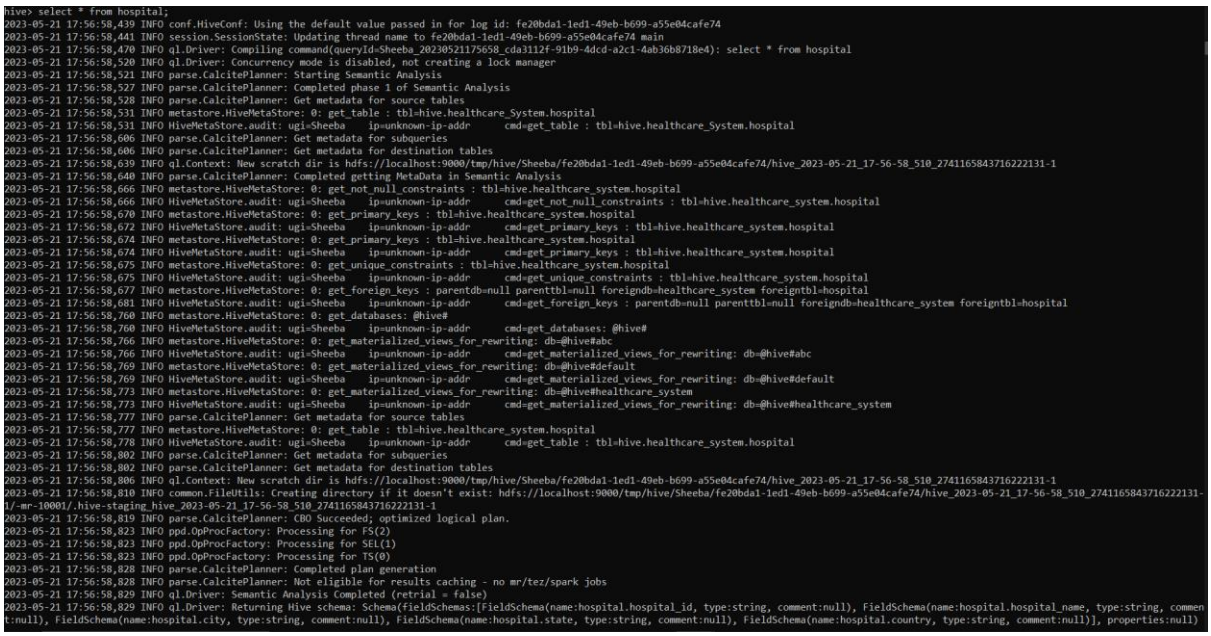
Showing imported tables in hive



Hive In web ui



Queries in hive



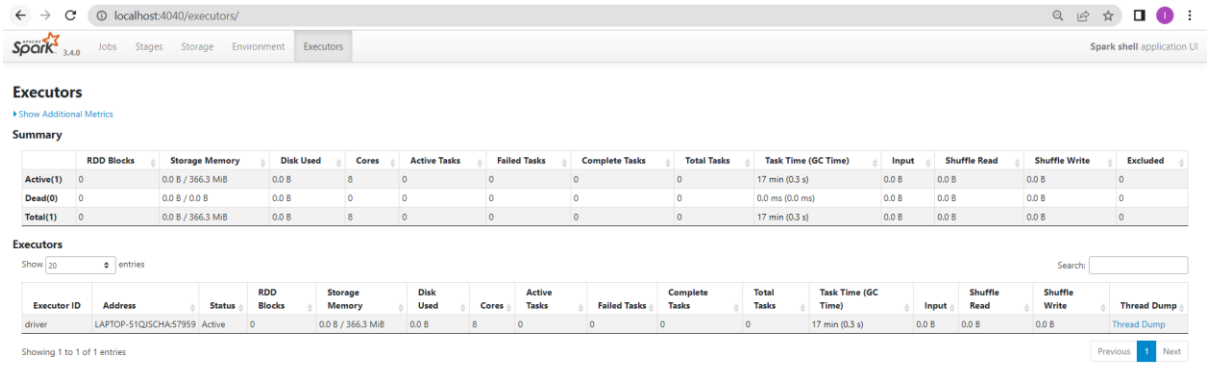
```
OK
2023-05-21 17:56:58,880 INFO ql.Driver: OK
2023-05-21 17:56:58,880 INFO ql.Driver: Concurrency mode is disabled, not creating a lock manager
2023-05-21 17:56:58,900 INFO mapred.FileInputFormat: Total input files to process : 1
2023-05-21 17:56:58,916 INFO lazy.LazyStruct: Missing fields! Expected 5 fields but only got 1! Ignoring similar problems.
2023-05-21 17:56:58,918 INFO exec.TableScanOperator: RECORDS_OUT_INTERMEDIATE:0, RECORDS_OUT_OPERATOR_TS_0:41,
2023-05-21 17:56:58,918 INFO exec.SelectOperator: RECORDS_OUT_INTERMEDIATE:0, RECORDS_OUT_OPERATOR_SEL_1:41,
2023-05-21 17:56:58,919 INFO exec.ListSinkOperator: RECORDS_OUT_INTERMEDIATE:0, RECORDS_OUT_OPERATOR_LIST_SINK_3:41,
H1000   All India Institute of Medical Sciences New Delhi       NaN     India
        NULL    NULL    NULL    NULL
H1001   Medanta The Medicity    Gurgaon Haryana India
        NULL    NULL    NULL    NULL
H1002   The Christian Medical College   Vellore Tamil Nadu      India
        NULL    NULL    NULL    NULL
H1003   PGIMER - Postgraduate Institute of Medical Education and Research       Chandigarh      Haryana India
        NULL    NULL    NULL    NULL
H1004   Apollo Hospital - Chennai       Chennai Tamil Nadu      India
        NULL    NULL    NULL    NULL
H1005   P. D. Hinduja National Hospital & Medical Research Centre       Mumbai  Maharashtra     India
        NULL    NULL    NULL    NULL
H1006   Breach Candy Hospital   Mumbai  Maharashtra     India
        NULL    NULL    NULL    NULL
H1007   Fortis Flt. Lt. Rajan Dhall Hospital    New Delhi       NaN     India
        NULL    NULL    NULL    NULL
H1008   King Edward Memorial Hospital   Mumbai  Maharashtra     India
        NULL    NULL    NULL    NULL
H1009   Indraprastha Apollo Hospital    Delhi   NaN     India
        NULL    NULL    NULL    NULL
H1010   Lilavati Hospital And Research Centre   Mumbai  Maharashtra     India
        NULL    NULL    NULL    NULL
H1011   Sir Ganga Ram Hospital  Delhi   NaN     India
        NULL    NULL    NULL    NULL
H1012   Bombay Hospital & Medical Research Centre       Mumbai  Maharashtra     India
        NULL    NULL    NULL    NULL
H1013   Apollo Health City - Jubilee Hills      Hyderabad       Telangana       India
        NULL    NULL    NULL    NULL
H1014   Fortis Hiranandani Hospital     Mumbai  Maharashtra     India
        NULL    NULL    NULL    NULL
H1015   Fortis Hospital Mulund  Mumbai  Maharashtra     India
        NULL    NULL    NULL    NULL
H1016   Jaslok Hospital and Research Centre     Mumbai  Maharashtra     India
        NULL    NULL    NULL    NULL
H1017   Manipal Hospitals       Bengaluru       Karnataka       India
        NULL    NULL    NULL    NULL
H1018   Yashoda Hospital Secunderabad   Hyderabad       Telangana       India
        NULL    NULL    NULL    NULL
H1019   Apollo Hospitals - Bannerghatta Road    Bengaluru       Karnataka       India
        NULL    NULL    NULL    NULL
Hospi   Hospital_name   city    state   countr
Time taken: 0.427 seconds, Fetched: 41 row(s)
2023-05-21 17:56:58,991 INFO CliDriver: Time taken: 0.427 seconds, Fetched: 41 row(s)
2023-05-21 17:56:58,991 INFO conf.HiveConf: Using the default value passed in for log id: fe20bda1-1ed1-49eb-b699-a55e04cafe74
```

```
hive> select * from hospital where hospital_id="H1013";
2023-05-21 17:59:31,775 INFO conf.HiveConf: Using the default value passed in for log id: fe20bda1-1ed1-49eb-b699-a55e04cafe74
2023-05-21 17:59:31,777 INFO session.SessionState: Updating thread name to fe20bda1-1ed1-49eb-b699-a55e04cafe74 main
2023-05-21 17:59:31,783 INFO ql.Driver: Compiling command(queryId=Sheeba_20230521175931_40f8255f-6901-4364-82f8-9163f5172aa8): select * from hospital where hospital_id="H1013"
2023-05-21 17:59:31,800 INFO ql.Driver: Concurrency mode is disabled, not creating a lock manager
2023-05-21 17:59:31,800 INFO parse.CalcitePlanner: Starting Semantic Analysis
2023-05-21 17:59:31,803 INFO parse.CalcitePlanner: Completed phase 1 of Semantic Analysis
2023-05-21 17:59:31,803 INFO parse.CalcitePlanner: Get metadata for source tables
2023-05-21 17:59:31,804 INFO metastore.HiveMetaStore: 0: get_table : tbl=hive.healthcare_System.hospital
2023-05-21 17:59:31,805 INFO HiveMetaStore.audit: ugi=Sheeba    ip=unknown-ip-addr      cmd=get_table : tbl=hive.healthcare_System.hospital
2023-05-21 17:59:31,829 INFO parse.CalcitePlanner: Get metadata for subqueries
2023-05-21 17:59:31,829 INFO parse.CalcitePlanner: Get metadata for destination tables
2023-05-21 17:59:31,838 INFO ql.Context: New scratch dir is hdfs://localhost:9000/tmp/hive/Sheeba/fe20bda1-1ed1-49eb-b699-a55e04cafe74/hive_2023-05-21_17-59-31_800_6168806748029619938-1
2023-05-21 17:59:31,839 INFO parse.CalcitePlanner: Completed getting MetaData in Semantic Analysis
2023-05-21 17:59:31,841 INFO metastore.HiveMetaStore: 0: get_not_null_constraints : tbl=hive.healthcare_system.hospital
2023-05-21 17:59:31,841 INFO HiveMetaStore.audit: ugi=Sheeba    ip=unknown-ip-addr      cmd=get_not_null_constraints : tbl=hive.healthcare_system.hospital
2023-05-21 17:59:31,842 INFO metastore.HiveMetaStore: 0: get_primary_keys : tbl=hive.healthcare_system.hospital
2023-05-21 17:59:31,843 INFO HiveMetaStore.audit: ugi=Sheeba    ip=unknown-ip-addr      cmd=get_primary_keys : tbl=hive.healthcare_system.hospital
2023-05-21 17:59:31,847 INFO metastore.HiveMetaStore: 0: get_primary_keys : tbl=hive.healthcare_system.hospital
2023-05-21 17:59:31,847 INFO HiveMetaStore.audit: ugi=Sheeba    ip=unknown-ip-addr      cmd=get_primary_keys : tbl=hive.healthcare_system.hospital
2023-05-21 17:59:31,859 INFO metastore.HiveMetaStore: 0: get_unique_constraints : tbl=hive.healthcare_system.hospital
2023-05-21 17:59:31,859 INFO HiveMetaStore.audit: ugi=Sheeba    ip=unknown-ip-addr      cmd=get_unique_constraints : tbl=hive.healthcare_system.hospital
2023-05-21 17:59:31,862 INFO metastore.HiveMetaStore: 0: get_foreign_keys : parentdb=null parenttbl=null foreigndb=healthcare_system foreigntbl=hospital
2023-05-21 17:59:31,863 INFO HiveMetaStore.audit: ugi=Sheeba    ip=unknown-ip-addr      cmd=get_foreign_keys : parentdb=null parenttbl=null foreigndb=healthcare_system foreigntbl=hospital
2023-05-21 17:59:32,030 INFO metastore.HiveMetaStore: 0: get_databases: @hive#
2023-05-21 17:59:32,030 INFO HiveMetaStore.audit: ugi=Sheeba    ip=unknown-ip-addr      cmd=get_databases: @hive#
2023-05-21 17:59:32,032 INFO metastore.HiveMetaStore: 0: get_materialized_views_for_rewriting: db=@hive#abc
2023-05-21 17:59:32,033 INFO HiveMetaStore.audit: ugi=Sheeba    ip=unknown-ip-addr      cmd=get_materialized_views_for_rewriting: db=@hive#abc
2023-05-21 17:59:32,037 INFO metastore.HiveMetaStore: 0: get_materialized_views_for_rewriting: db=@hive#default
2023-05-21 17:59:32,038 INFO HiveMetaStore.audit: ugi=Sheeba    ip=unknown-ip-addr      cmd=get_materialized_views_for_rewriting: db=@hive#default
2023-05-21 17:59:32,039 INFO metastore.HiveMetaStore: 0: get_materialized_views_for_rewriting: db=@hive#healthcare_system
2023-05-21 17:59:32,041 INFO HiveMetaStore.audit: ugi=Sheeba    ip=unknown-ip-addr      cmd=get_materialized_views_for_rewriting: db=@hive#healthcare_system
2023-05-21 17:59:32,049 INFO parse.CalcitePlanner: Get metadata for source tables
2023-05-21 17:59:32,050 INFO metastore.HiveMetaStore: 0: get_table : tbl=hive.healthcare_system.hospital
2023-05-21 17:59:32,050 INFO HiveMetaStore.audit: ugi=Sheeba    ip=unknown-ip-addr      cmd=get_table : tbl=hive.healthcare_system.hospital
2023-05-21 17:59:32,075 INFO parse.CalcitePlanner: Get metadata for subqueries
2023-05-21 17:59:32,075 INFO parse.CalcitePlanner: Get metadata for destination tables
2023-05-21 17:59:32,077 INFO ql.Context: New scratch dir is hdfs://localhost:9000/tmp/hive/Sheeba/fe20bda1-1ed1-49eb-b699-a55e04cafe74/hive_2023-05-21_17-59-31_800_6168806748029619938-1
2023-05-21 17:59:32,081 INFO common.FileUtils: Creating directory if it doesn't exist: hdfs://localhost:9000/tmp/hive/Sheeba/fe20bda1-1ed1-49eb-b699-a55e04cafe74/hive_2023-05-21_17-59-31_800_6168806748029619938-
1/-mr-10001/.hive-staging_hive_2023-05-21_17-59-31_800_6168806748029619938-1
2023-05-21 17:59:32,083 INFO parse.CalcitePlanner: CBO Succeeded; optimized logical plan.
2023-05-21 17:59:32,102 INFO ppd.OpProcFactory: Processing for FS(3)
2023-05-21 17:59:32,102 INFO ppd.OpProcFactory: Processing for SEL(2)
2023-05-21 17:59:32,102 INFO ppd.OpProcFactory: Processing for FIL(1)
2023-05-21 17:59:32,114 INFO ppd.OpProcFactory: Processing for TS(0)
2023-05-21 17:59:32,196 INFO parse.CalcitePlanner: Completed plan generation
2023-05-21 17:59:32,196 INFO parse.CalcitePlanner: Not eligible for results caching - no mr/tez/spark jobs
2023-05-21 17:59:32,197 INFO ql.Driver: Semantic Analysis Completed (retrial = false)
2023-05-21 17:59:32,197 INFO ql.Driver: Returning Hive schema: Schema(fieldSchemas:[FieldSchema(name:hospital.hospital_id, type:string, comment:null), FieldSchema(name:hospital.hospital_name, type:string, commen
t:null), FieldSchema(name:hospital.city, type:string, comment:null), FieldSchema(name:hospital.state, type:string, comment:null), FieldSchema(name:hospital.country, type:string, comment:null)], properties:null)
2023-05-21 17:59:32,201 INFO exec.TableScanOperator: Initializing operator TS[0]
```

```
2023-05-21 17:59:32,217 INFO ql.Driver: Executing command(queryId=Sheeba_20230521175931_40f8255f-6901-4364-82f8-9163f5172aa8): select * from hospital where hospital_id="H1013"
2023-05-21 17:59:32,221 INFO ql.Driver: Completed executing command(queryId=Sheeba_20230521175931_40f8255f-6901-4364-82f8-9163f5172aa8); Time taken: 0.004 seconds
OK
2023-05-21 17:59:32,228 INFO ql.Driver: OK
2023-05-21 17:59:32,228 INFO ql.Driver: Concurrency mode is disabled, not creating a lock manager
2023-05-21 17:59:32,234 INFO mapred.FileInputFormat: Total input files to process : 1
2023-05-21 17:59:32,258 INFO lazy.LazyStruct: Missing fields! Expected 5 fields but only got 1! Ignoring similar problems.
2023-05-21 17:59:32,259 INFO exec.TableScanOperator: RECORDS_OUT_INTERMEDIATE:0, RECORDS_OUT_OPERATOR_TS_0:41,
2023-05-21 17:59:32,264 INFO exec.FilterOperator: RECORDS_OUT_INTERMEDIATE:0, RECORDS_OUT_OPERATOR_FIL_4:1,
2023-05-21 17:59:32,264 INFO exec.SelectOperator: RECORDS_OUT_OPERATOR_SEL_2:1, RECORDS_OUT_INTERMEDIATE:0,
2023-05-21 17:59:32,265 INFO exec.ListSinkOperator: RECORDS_OUT_OPERATOR_LIST_SINK_5:1, RECORDS_OUT_INTERMEDIATE:0,
H1013   Apollo Health City - Jubilee Hills      Hyderabad       Telangana       India
Time taken: 0.449 seconds, Fetched: 1 row(s)
2023-05-21 17:59:32,271 INFO CliDriver: Time taken: 0.449 seconds, Fetched: 1 row(s)
2023-05-21 17:59:32,272 INFO conf.HiveConf: Using the default value passed in for log id: fe20bda1-1ed1-49eb-b699-a55e04cafe74
2023-05-21 17:59:32,272 INFO session.SessionState: Resetting thread name to  main
hive> 2023-05-21 17:59:34,293 INFO metastore.HiveMetaStore: 2: Opening raw store with implementation class:org.apache.hadoop.hive.metastore.ObjectStore
2023-05-21 17:59:34,295 INFO metastore.ObjectStore: ObjectStore, initialize called
2023-05-21 17:59:34,300 INFO metastore.MetaStoreDirectSql: Using direct SQL, underlying DB is DERBY
2023-05-21 17:59:34,300 INFO metastore.ObjectStore: Initialized ObjectStore
```

Spark is installed

```
Administrator: Command Prompt - spark-shell
04/07/2023  08:13 AM             1,935 find-spark-home
04/07/2023  08:13 AM             2,685 find-spark-home.cmd
04/07/2023  08:13 AM             2,337 load-spark-env.cmd
04/07/2023  08:13 AM             2,678 load-spark-env.sh
04/07/2023  08:13 AM             2,636 pyspark
04/07/2023  08:13 AM             1,170 pyspark.cmd
04/07/2023  08:13 AM             1,542 pyspark2.cmd
04/07/2023  08:13 AM             1,030 run-example
04/07/2023  08:13 AM             1,223 run-example.cmd
04/07/2023  08:13 AM             3,561 spark-class
04/07/2023  08:13 AM             1,180 spark-class.cmd
04/07/2023  08:13 AM             2,891 spark-class2.cmd
04/07/2023  08:13 AM             1,154 spark-connect-shell
04/07/2023  08:13 AM             3,122 spark-shell
04/07/2023  08:13 AM             1,178 spark-shell.cmd
04/07/2023  08:13 AM             1,818 spark-shell2.cmd
04/07/2023  08:13 AM             1,065 spark-sql
04/07/2023  08:13 AM             1,173 spark-sql.cmd
04/07/2023  08:13 AM             1,118 spark-sql2.cmd
04/07/2023  08:13 AM             1,040 spark-submit
04/07/2023  08:13 AM             1,180 spark-submit.cmd
04/07/2023  08:13 AM             1,155 spark-submit2.cmd
04/07/2023  08:13 AM             1,047 sparkR
04/07/2023  08:13 AM             1,168 sparkR.cmd
04/07/2023  08:13 AM             1,097 sparkR2.cmd
              28 File(s)         55,323 bytes
               2 Dir(s)  227,609,923,584 bytes free

C:\spark\spark-3.4.0-bin-hadoop3\bin>spark-shell
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Spark context Web UI available at http://LAPTOP-51QJSCHA:4040
Spark context available as 'sc' (master = local[*], app id = local-1684683448971).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.4.0
      /_/

Using Scala version 2.12.17 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_351)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```



Data Analysis using Spark

```
pip install pyspark
```

```
Requirement already satisfied: pyspark in c:\users\sheeba\anaconda3new\lib\site-packages (3.4.0)
Requirement already satisfied: py4j==0.10.9.7 in c:\users\sheeba\anaconda3new\lib\site-packages (from pyspark) (0.10.9.7)
Note: you may need to restart the kernel to use updated packages.
```

```
from pyspark.sql import SparkSession
```

```
spark = SparkSession.builder \
    .appName("Spark Hive Example") \
    .config("spark.sql.warehouse.dir", "hdfs://localhost:9870/user/hive/warehouse") \
    .config("spark.sql.catalogImplementation", "hive") \
    .enableHiveSupport() \
    .getOrCreate()
```

```
# USe healthcare_system Database
spark.sql("use healthcare_system").show()
```

```
++
||
++
++
```

```
# Print all the tables which are present in the healthcare_system database.
sparkdf = spark.sql("show tables")
sparkdf.toPandas().to_csv('Spark Outputs for Visualization/Database.csv')
sparkdf.show()
```

```
+----------------+----------+-----------+
|        database| tableName|isTemporary|
+----------------+----------+-----------+
|healthcare_system|    claims|      false|
|healthcare_system|   disease|      false|
|healthcare_system|    groups|      false|
|healthcare_system|grp_subgrp|      false|
|healthcare_system|  hospital|      false|
|healthcare_system|   patient|      false|
|healthcare_system|  subgroup|      false|
|healthcare_system|subscriber|      false|
+----------------+----------+-----------+
```

```
# Describe the claims table
sparkdf = spark.sql("desc claims")
sparkdf.toPandas().to_csv('Spark Outputs for Visualization/Claims_tble.csv')
sparkdf.show()
```

```
+----------------+---------+-------+
|        col_name|data_type|comment|
+----------------+---------+-------+
|        claim_id|      int|   null|
|      patient_id|      int|   null|
|    disease_name|   string|   null|
|          sub_id|   string|   null|
|claim_or_rejected|   string|   null|
|      claim_type|   string|   null|
|    claim_amount|   double|   null|
|      claim_date|   string|   null|
+----------------+---------+-------+
```

```
# Describe the disease table
sparkdf = spark.sql("desc disease")
sparkdf.toPandas().to_csv('Spark Outputs for Visualization/disease_tble.csv')
sparkdf.show()
```

```
+------------+---------+-------+
|    col_name|data_type|comment|
+------------+---------+-------+
|  disease_id|      int|   null|
|disease_name|   string|   null|
|   subgrp_id|   string|   null|
+------------+---------+-------+
```

```
# Describe the groups table
sparkdf = spark.sql("desc groups")
sparkdf.toPandas().to_csv('Spark Outputs for Visualization/groups_tble.csv')
sparkdf.show()
```

```
+---------------+---------+-------+
|       col_name|data_type|comment|
+---------------+---------+-------+
|         grp_sk|      int|   null|
|         grp_id|   string|   null|
|       grp_name|   string|   null|
|premium_written|      int|   null|
|           city|   string|   null|
|       zip_code|      int|   null|
|        country|   string|   null|
|       grp_type|   string|   null|
+---------------+---------+-------+
```

```
# Describe the grp_subgrp table
sparkdf = spark.sql("desc grp_subgrp")
sparkdf.toPandas().to_csv('Spark Outputs for Visualization/grp_subgrp_tble.csv')
sparkdf.show()
```

```
+---------+---------+-------+
| col_name|data_type|comment|
+---------+---------+-------+
|grpsub_sk|      int|   null|
|     g_id|   string|   null|
|     s_id|   string|   null|
+---------+---------+-------+
```

```
# Describe the hospital table
sparkdf = spark.sql("desc hospital")
sparkdf.toPandas().to_csv('Spark Outputs for Visualization/hospital_tble.csv')
sparkdf.show()
```

```
+-------------+---------+-------+
|     col_name|data_type|comment|
+-------------+---------+-------+
|  hospital_id|   string|   null|
|hospital_name|   string|   null|
|         city|   string|   null|
|        state|   string|   null|
|      country|   string|   null|
+-------------+---------+-------+
```

```
# Describe the patient table
sparkdf = spark.sql("desc patient")
sparkdf.toPandas().to_csv('Spark Outputs for Visualization/patient_tble.csv')
sparkdf.show()
```

```
+------------------+---------+-------+
|          col_name|data_type|comment|
+------------------+---------+-------+
|        patient_id|      int|   null|
|      patient_name|   string|   null|
|    patient_gender|   string|   null|
|patient_birth_date|   string|   null|
|     patient_phone|   string|   null|
|      disease_name|   string|   null|
|              city|   string|   null|
|       hospital_id|   string|   null|
+------------------+---------+-------+
```

```
# Describe the subgroup table
sparkdf = spark.sql("desc subgroup")
sparkdf.toPandas().to_csv('Spark Outputs for Visualization/subgroup_tble.csv')
sparkdf.show()
```

```
+---------------+---------+-------+
|       col_name|data_type|comment|
+---------------+---------+-------+
|      subgrp_sk|      int|   null|
|      subgrp_id|   string|   null|
|    subgrp_name|   string|   null|
|monthly_premium|   double|   null|
+---------------+---------+-------+
```

```
# Describe the subscriber table
sparkdf = spark.sql("desc subscriber")
sparkdf.toPandas().to_csv('Spark Outputs for Visualization/subscriber_tble.csv')
sparkdf.show()
```

```
+----------+---------+-------+
|  col_name|data_type|comment|
+----------+---------+-------+
|    sub_id|   string|   null|
|first_name|   string|   null|
| last_name|   string|   null|
|    street|   string|   null|
|birth_date|   string|   null|
|    gender|   string|   null|
|     phone|   string|   null|
|      city|   string|   null|
|  zip_code|      int|   null|
|   country|   string|   null|
| subgrp_id|   string|   null|
|  elig_ind|   string|   null|
|  eff_date|   string|   null|
| term_date|   string|   null|
+----------+---------+-------+
```

```
# which disease having maximum number of claims.
results = spark.sql("select disease_name,count(claim_id) as \
                    max from claims group by disease_name order by max desc")
```

```
results.toPandas().to_csv('Spark Outputs for Visualization/query1.csv')
results.show()
```


```
# which disease having maximum number of claims.
results = spark.sql("select disease_name,count(claim_id) as \
                    max from claims group by disease_name order by max desc")
```

```
results.toPandas().to_csv('Spark Outputs for Visualization/query1.csv')
results.show()
```


```
+----------------+---+
|    disease_name|max|
+----------------+---+
|        Glaucoma|  3|
| Phenylketonuria|  3|
|    Galactosemia|  3|
|     Pet allergy|  3|
|         Anthrax|  3|
|    Head banging|  3|
|          Asthma|  2|
|          Scurvy|  2|
|Drug consumption|  2|
|  Bladder cancer|  2|
|   Rett Syndrome|  2|
|         Choking|  2|
|             Flu|  2|
|      Lymphedema|  2|
|    Mold allergy|  2|
| Fanconi anaemia|  2|
|         Measles|  2|
|          Stroke|  2|
|         Cholera|  2|
|         Malaria|  2|
+----------------+---+
only showing top 20 rows
```

```
# Find those Subscribers having age less than 30 and they subscribe any subgroup
subscriber_ages = spark.sql("select birth_date,current_date() as CurrentDate,\
          year(current_date())-year(birth_date) as age from subscriber")
```

```
res = subscriber_ages.filter(subscriber_ages.age < 30).count()
```

```
print("Subscribers having age less than 30 --> ",res)
```

```
Subscribers having age less than 30 -->  4
```

```
# Find out which group has maximum subgroups.
sparkdf = spark.sql("select g_id,count(s_id) as max from grp_subgrp group by g_id order by max desc")
sparkdf.toPandas().to_csv('Spark Outputs for Visualization/query3.csv')
sparkdf.show()
```

```
+------+---+
|  g_id|max|
+------+---+
|GRP104|  2|
|GRP147|  2|
|GRP143|  2|
|GRP108|  1|
|GRP105|  1|
|GRP101|  1|
|GRP148|  1|
|GRP102|  1|
|GRP103|  1|
|GRP114|  1|
|GRP142|  1|
|GRP133|  1|
|GRP112|  1|
|GRP122|  1|
|GRP138|  1|
|GRP157|  1|
|GRP127|  1|
|GRP123|  1|
|GRP126|  1|
|GRP110|  1|
+------+---+
only showing top 20 rows
```

```
# Find out hospital which serve most number of patients
sparkdf = spark.sql("select hospital_name,count(patient_id) as total_patient \
          from hospital join patient on hospital.hospital_id=patient.hospital_id \
          group by hospital_name order by total_patient desc")

sparkdf.toPandas().to_csv('Spark Outputs for Visualization/query4.csv')
sparkdf.show()
```

```
+--------------------+-------------+
|       hospital_name|total_patient|
+--------------------+-------------+
|   Manipal Hospitals|            9|
|Apollo Hospitals ...|            8|
|Medanta The Medicity|            7|
|Jaslok Hospital a...|            6|
|Indraprastha Apol...|            5|
|PGIMER - Postgrad...|            4|
|Apollo Hospital -...|            4|
|Fortis Hospital M...|            4|
|King Edward Memor...|            3|
|Apollo Health Cit...|            3|
|Yashoda Hospital ...|            3|
|Bombay Hospital &...|            3|
|Fortis Hiranandan...|            2|
|Lilavati Hospital...|            2|
|The Christian Med...|            2|
|Fortis Flt. Lt. R...|            1|
|P. D. Hinduja Nat...|            1|
|Breach Candy Hosp...|            1|
|All India Institu...|            1|
|Sir Ganga Ram Hos...|            1|
+--------------------+-------------+
```

```
# Find out which subgroups subscribe most number of times
sparkdf = spark.sql("select subgrp_name, \
          count(sub_id) as cunt from subgroup join \
          subscriber on subgroup.subgrp_id = subscriber.subgrp_id  group by subgrp_name order by cunt desc")

sparkdf.toPandas().to_csv('Spark Outputs for Visualization/query5.csv')
sparkdf.show()
```

```
+------------------+----+
|       subgrp_name|cunt|
+------------------+----+
|           Therapy|  14|
|        Hereditary|  11|
|             Viral|  11|
|Deficiency Diseases|  11|
|         Physiology|  10|
|          Allergies|  10|
|           Accident|  10|
|             Cancer|   9|
|    Self inflicted|   7|
| Infectious disease|   7|
+------------------+----+
```

```
# Find out total number of claims which were rejected
sparkdf = spark.sql("select claim_or_rejected,count(claim_id) from claims group by claim_or_rejected")

sparkdf.toPandas().to_csv('Spark Outputs for Visualization/query6.csv')
sparkdf.show()
print("The above Result shows the total 52 claims were rejected")
```

```
+----------------+---------------+
|claim_or_rejected|count(claim_id)|
+----------------+---------------+
|               Y|             18|
|               N|             52|
+----------------+---------------+

The above Result shows the total 52 claims were rejected
```

```
# From where most claims are comming (city)
sparkdf = spark.sql("select patient.city,count(claim_id) as maxclaim from patient join claims on \
          claims.patient_id = patient.patient_id group by patient.city order by maxclaim desc limit 10")
sparkdf.toPandas().to_csv('Spark Outputs for Visualization/query7.csv')
sparkdf.show()
```

```
+-----------+--------+
|       city|maxclaim|
+-----------+--------+
|     Mysore|       2|
|   Amravati|       2|
|  Kamarhati|       2|
|   Jabalpur|       2|
|Bihar Sharif|       2|
|  Ghaziabad|       2|
|      Morbi|       2|
| Karimnagar|       2|
|  Bangalore|       1|
|    Udaipur|       1|
+-----------+--------+
```

```
# Which groups of policies subscriber subscribe mostly Goverment or private
sparkdf = spark.sql("select grp_type,count(grp_id) from groups group by grp_type")
sparkdf.toPandas().to_csv('Spark Outputs for Visualization/query8.csv')
sparkdf.show()
```

```
+--------+-------------+
|grp_type|count(grp_id)|
+--------+-------------+
|  Govt.|            7|
| Private|           51|
+--------+-------------+
```

```
res = spark.sql("select grp_subgrp.g_id from subscriber \
          join grp_subgrp on grp_subgrp.s_id = subscriber.subgrp_id")

res.registerTempTable("grp_tble")

sparkdf = spark.sql("select grp_type,count(grp_tble.g_id) from groups \
          join grp_tble on groups.grp_id = grp_tble.g_id group by grp_type")

sparkdf.toPandas().to_csv('Spark Outputs for Visualization/query9.csv')
sparkdf.show()
```

```
+--------+-----------+
|grp_type|count(g_id)|
+--------+-----------+
|   Govt.|         35|
| Private|        347|
+--------+-----------+
```

```
print("The Above Result shows that most of the subscribers subscribe private groups")
```

```
The Above Result shows that most of the subscribers subscribe private groups
```

```
# Average monthly premium subscriber pay to insurance company subgroup.
sparkdf = spark.sql("select subgroup.subgrp_id,avg(monthly_premium) from subscriber right join \
          subgroup on subscriber.subgrp_id = subgroup.subgrp_id group by \
          subgroup.subgrp_id order by subgroup.subgrp_id")
sparkdf.toPandas().to_csv('Spark Outputs for Visualization/query10.csv')
sparkdf.show()
```

```
+---------+--------------------+
|subgrp_id|avg(monthly_premium)|
+---------+--------------------+
|     5101|              3000.0|
|     5102|              1000.0|
|     5103|              2000.0|
|     5104|              1500.0|
|     5105|              2300.0|
|     5106|              1200.0|
|     5107|              3200.0|
|     5108|              1500.0|
|     5109|              2000.0|
|     5110|              1000.0|
+---------+--------------------+
```

```
# Find out Which group is most profitable
sparkdf = spark.sql("select grp_id,premium_written from groups order by premium_written desc")
sparkdf.toPandas().to_csv('Spark Outputs for Visualization/query11.csv')
sparkdf.show()
```

```
+------+---------------+
|grp_id|premium_written|
+------+---------------+
|GRP147|          99000|
|GRP131|          99000|
|GRP123|          99000|
|GRP118|          97000|
|GRP154|          95000|
|GRP133|          93000|
|GRP157|          92000|
|GRP134|          90000|
|GRP143|          90000|
|GRP106|          89000|
|GRP129|          88000|
|GRP152|          87000|
|GRP153|          86000|
|GRP150|          84000|
|GRP124|          81000|
|GRP122|          79000|
|GRP115|          79000|
|GRP121|          78000|
|GRP109|          78000|
|GRP101|          72000|
+------+---------------+
only showing top 20 rows
```

```
# List all the patients below age of 18 who admit for cancer

sparkdf = spark.sql("select patient_id, patient_name,\
          year(current_date())-year(patient_birth_date) as age from patient \
          where disease_name like '%cancer' order by age limit 2")
sparkdf.toPandas().to_csv('Spark Outputs for Visualization/query12.csv')
sparkdf.show()
```

```
+----------+------------+---+
|patient_id|patient_name|age|
+----------+------------+---+
|    194166|          NA|  9|
|    197441|    Deependu| 18|
+----------+------------+---+
```

```
# List patients who have cashless insura0nce and have total charges greater than or equal for Rs. 50,000.

sparkdf = spark.sql("select patient_name,patient_gender,patient_birth_date \
          from patient join claims on patient.patient_id = claims.patient_id \
          where claim_amount >= 50000 and claim_type = 'claims of value'")
sparkdf.toPandas().to_csv('Spark Outputs for Visualization/query13.csv')
sparkdf.show()
```

```
+------------+--------------+------------------+
|patient_name|patient_gender|patient_birth_date|
+------------+--------------+------------------+
|   Anjushree|          Male|        1982-06-28|
|  Chitranjan|        Female|        2020-10-27|
|      Gensho|          Male|        1991-07-27|
|  Vaijayanti|          Male|        1969-04-06|
|       Aakar|        Female|        1990-04-24|
|          NA|        Female|        1959-01-06|
|       Saroj|        Female|        1953-07-21|
|     Bhagvan|        Female|        2011-02-26|
|  Dharmadaas|          Male|        1964-10-25|
|       Umang|        Female|        2017-02-26|
|          NA|          Male|        1955-06-03|
|      Kishan|          Male|        1955-06-30|
|          NA|        Female|        1953-04-04|
|     Devnath|        Female|        1982-02-22|
|      Harbir|        Female|        1960-02-24|
|       Ekant|          Male|        1969-11-01|
|          NA|          Male|        2013-10-30|
|          NA|          Male|        1956-04-04|
|       Lalit|        Female|        1978-04-30|
|     Ujjawal|          Male|        1965-12-31|
+------------+--------------+------------------+
```

```
# List female patients over the age of 40 that have undergone knee surgery in the past year

sparkdf = spark.sql("select patient_name from patient where patient_gender = 'Female' and disease_name = 'Heart Attack'")
sparkdf.toPandas().to_csv('Spark Outputs for Visualization/query14.csv')
sparkdf.show()
```

```
+------------+
|patient_name|
+------------+
|     Upasana|
+------------+
```

Spark outputs:

Claims_tbl.csv

Database.csv

disease_tbl.csv

groups_tbl.csv

grp_subgrp_tble.csv

hospital_tbl.csv

patient_tbl.csv

query1.csv

query2.csv

query3.csv

query4.csv

query5.csv

query6.csv

query7.csv

query8.csv

query9.csv

query10.csv

query11.csv

query12.csv

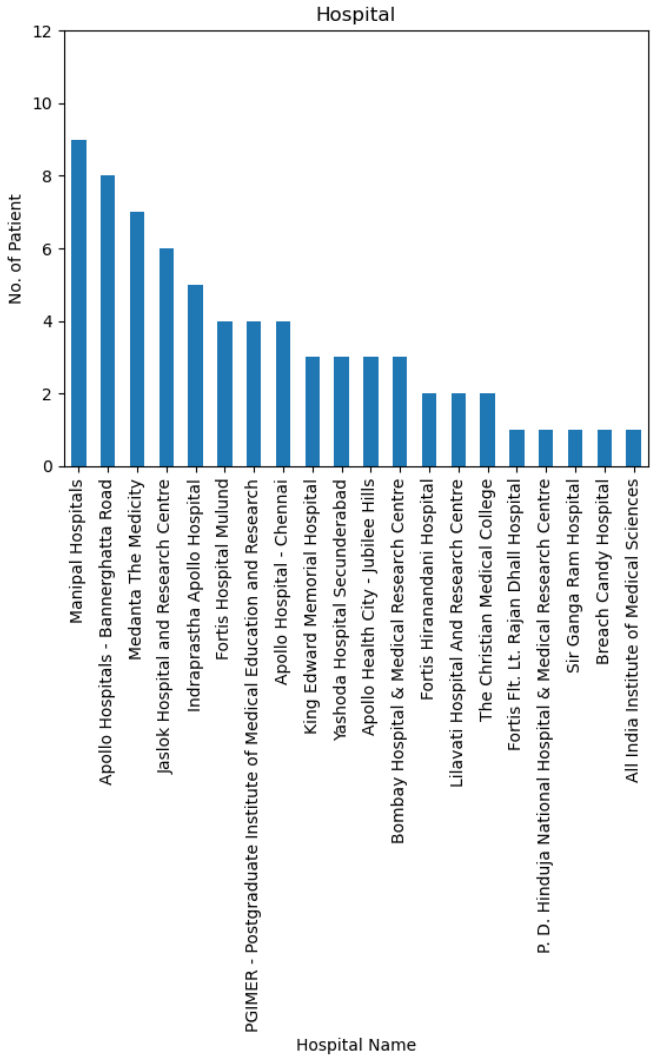
query13.csv

Visualization:

In [1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:
```
# Find out hospital which serve most number of patients

df=pd.read_csv('query4.csv')
```

In [3]:
```
df.plot(x='hospital_name',y='total_patient',kind='bar',legend=None)
plt.title("Hospital")
plt.ylabel('No. of Patient')
plt.xlabel("Hospital Name")
plt.ylim(ymin=0,ymax=12)
plt.savefig('Hospital with most patient.png',bbox_inches='tight')
```


Hospital
No. of Patient
0
2
4
6
8
10
12
Manipal Hospitals
Apollo Hospitals - Bannerghatta Road
Medanta The Medicity
Jaslok Hospital and Research Centre
Indraprastha Apollo Hospital
Fortis Hospital Mulund
PGIMER - Postgraduate Institute of Medical Education and Research
Apollo Hospital - Chennai
King Edward Memorial Hospital
Yashoda Hospital Secunderabad
Apollo Health City - Jubilee Hills
Bombay Hospital & Medical Research Centre
Fortis Hiranandani Hospital
Lilavati Hospital And Research Centre
The Christian Medical College
Fortis Flt. Lt. Rajan Dhall Hospital
P. D. Hinduja National Hospital & Medical Research Centre
Sir Ganga Ram Hospital
Breach Candy Hospital
All India Institute of Medical Sciences
Hospital Name

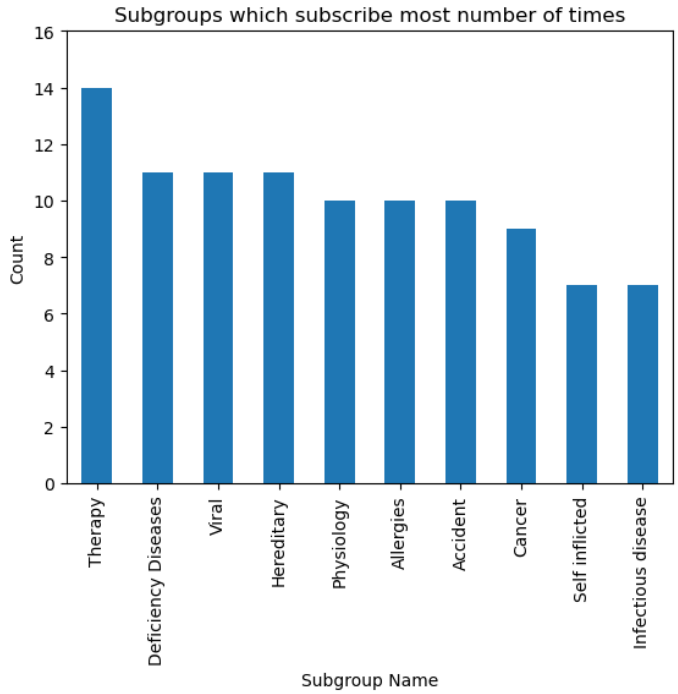
In [4]:
```
## Find out which subgroups subscribe most number of times

df1=pd.read_csv('query5.csv')
```

In [5]:
```
df1
```

Out[5]:

	Unnamed: 0	subgrp_name	cunt
0	0	Therapy	14
1	1	Deficiency Diseases	11
2	2	Viral	11
3	3	Hereditary	11
4	4	Physiology	10
5	5	Allergies	10
6	6	Accident	10
7	7	Cancer	9
8	8	Self inflicted	7
9	9	Infectious disease	7

In [6]:
```
df1.plot(x='subgrp_name',y='cunt',kind='bar',legend=None)
plt.title("Subgroups which subscribe most number of times")
plt.ylabel('Count')
plt.xlabel("Subgroup Name")
plt.ylim(ymin=0,ymax=16)
plt.savefig('Subgroups which subscribe most number of times',bbox_inches='tight')
```



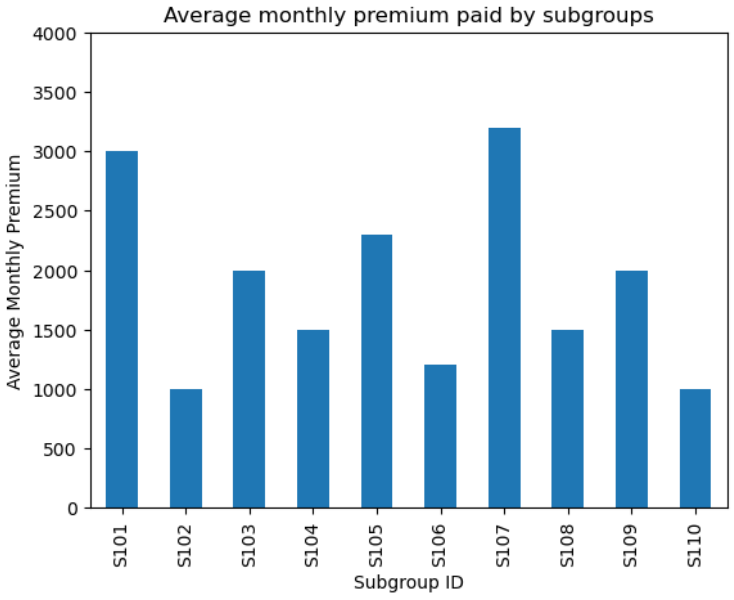
In [7]:
```
# Average monthly premium subscriber pay to insurance company subgroup.

df2=pd.read_csv('query10.csv')
```

In [8]:
```
df2
```

Out[8]:

	Unnamed: 0	subgrp_id	avg(monthly_premium)
0	0	S101	3000.0
1	1	S102	1000.0
2	2	S103	2000.0
3	3	S104	1500.0
4	4	S105	2300.0
5	5	S106	1200.0
6	6	S107	3200.0
7	7	S108	1500.0
8	8	S109	2000.0
9	9	S110	1000.0

In [9]:
```
df2.plot(x='subgrp_id',y='avg(monthly_premium)',kind='bar',legend=None)
plt.title("Average monthly premium paid by subgroups")
plt.ylabel('Average Monthly Premium')
plt.xlabel("Subgroup ID")
plt.ylim(ymin=0,ymax=4000)
plt.savefig('Average Monthly Premium Paid by Subgroups',bbox_inches='tight')
```



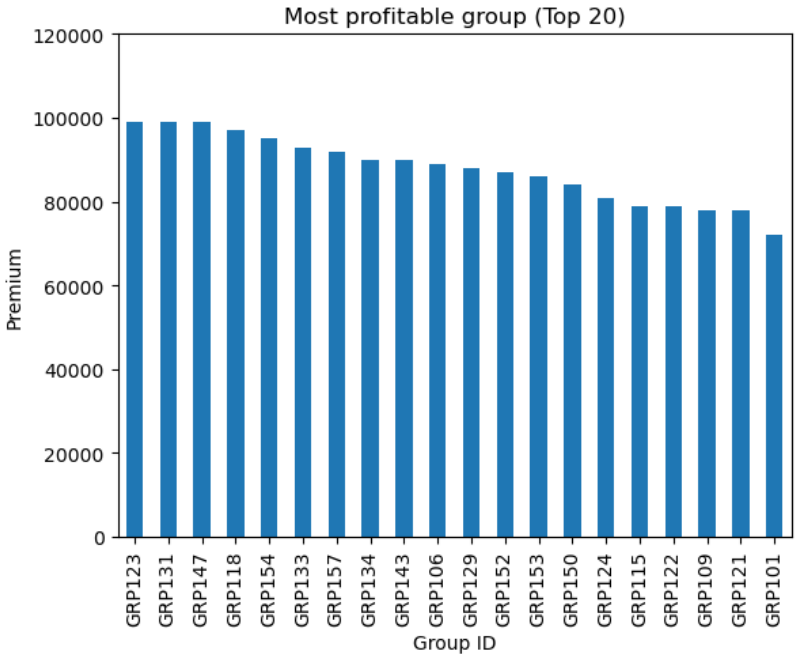
In [10]:
```
# Find out Which group is most profitable

df3=pd.read_csv('query11.csv')
```

In [11]:
```
df3.head()
```

Out[11]:

	Unnamed: 0	grp_id	premium_written
0	0	GRP123	99000
1	1	GRP131	99000
2	2	GRP147	99000
3	3	GRP118	97000
4	4	GRP154	95000

In [12]:
```
df3.head(n=20).plot(x='grp_id',y='premium_written',kind='bar',legend=None)
plt.title("Most profitable group (Top 20)")
plt.xlabel("Group ID")
plt.ylabel("Premium")
plt.ylim(ymin=0,ymax=120000)
plt.savefig('Most Profitable Groups',bbox_inches='tight')
```



In [13]:
```
import matplotlib.pyplot as plt
from importlib import reload
plt=reload(plt)
```

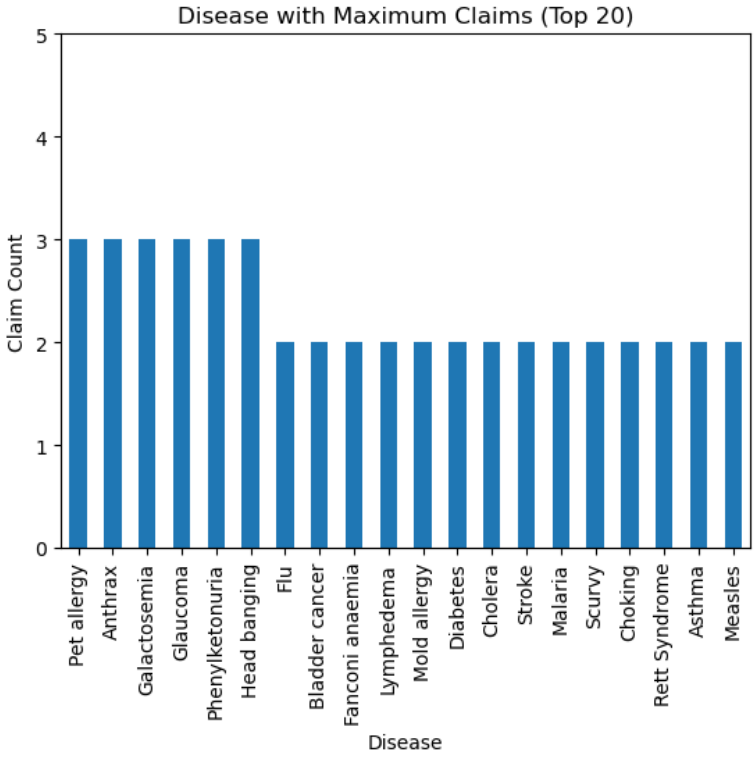
In [14]:
```
# which disease having maximum number of claims.

df4=pd.read_csv('query1.csv')
```

In [15]:
```
df4.head()
```

Out[15]:

	Unnamed: 0	disease_name	max
0	0	Pet allergy	3
1	1	Anthrax	3
2	2	Galactosemia	3
3	3	Glaucoma	3
4	4	Phenylketonuria	3

In [16]:
```
df4.head(n=20).plot(x='disease_name',y='max',kind='bar',legend=None)
plt.title("Disease with Maximum Claims (Top 20)")
plt.xlabel("Disease")
plt.ylabel("Claim Count")
plt.ylim(ymin=0,ymax=5)
plt.savefig('Disease with max Claim',bbox_inches='tight')
```



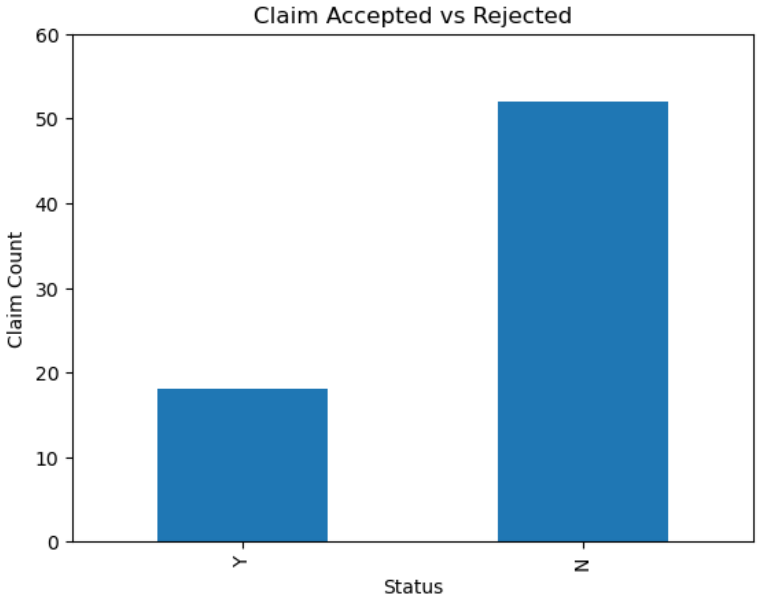
In [17]:
```
# Find out total number of claims which were rejected

df5=pd.read_csv('query6.csv')
```

In [18]:
```
df5
```

Out[18]:

	Unnamed: 0	claim_or_rejected	count(claim_id)
0	0	Y	18
1	1	N	52

In [19]:
```
df5.head(n=20).plot(x='claim_or_rejected',y='count(claim_id)',kind='bar',legend=None)
plt.title("Claim Accepted vs Rejected")
plt.xlabel("Status")
plt.ylabel("Claim Count")
plt.ylim(ymin=0,ymax=60)
plt.savefig('Claim Accepted vs Rejected',bbox_inches='tight')
```


Claim Accepted vs Rejected
Claim Count
60
50
40
30
20
10
0
Y
N
Status

CHAPTER 6

CONCLUSION AND FUTURE WORKS

6.1 CONCLUSION

Implementing a big data analytics system in a healthcare insurance company offers tremendous potential for revenue growth and enhanced customer understanding. By addressing challenges such as data volume management, security, integration, analysis complexity, scalability, talent shortage, and ethical considerations, the project can pave the way for personalized insurance policy offerings, accurate royalty calculations, and improved service provision. Leveraging the power of big data analytics, the system enables the company to track customer behavior and conditions, leading to tailored insurance solutions and increased revenue. Additionally, the project promotes better decision-making through advanced analytics techniques, while adhering to data privacy regulations and ethical standards. Ultimately, the successful implementation of the project contributes to a better understanding of customers and an improved overall customer experience in the healthcare insurance industry.Some of our analysis includes average monthly premium for each subgroup,which disease have maximum number of claims,which group is most profitable etc.

6.2 FUTURE WORK

In this part of the implementation, the data files of JSON and csv formats are prepared and preprocessed. An RDBMS model is created for storage of contents of these data files which will be used for further processing and analysis in subsequent steps using technologies such as Sqoop, HDFS and Hive. This will enable us to analyze the data from a healthcare insurance company to obtain useful results and test it to increase the revenue of our company.

REFERENCES

[1]Sheeran, M., & Steele, R. (2017, October). A framework for big data technology in health and healthcare. In 2017 IEEE 8th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON) (pp. 401-407). IEEE.

[2]Zheng, L., & Guo, L. (2020, April). Application of big data technology in insurance innovation. In International conference on education, economics and information management (ICEEIM 2019) (pp. 285-294). Atlantis Press.