# An Analysis of COVID-19 Pandemic and Vaccination Progress in Europe: A Data Warehousing Approach

Xhesika Feto

# An Analysis of COVID-19 Pandemic and Vaccination Progress in Europe: A Data Warehousing Approach

## I. Introduction

The COVID-19 epidemic has had a profound effect on society and altered how people live and work. It is crucial to have accurate and current information to improve public health strategies and decision-making as countries continue to fight this infection. Using data from the European Centre for Disease Prevention and Control, this paper analyses the COVID-19 status in Europe. (ECDC). Three key datasets provided by the ECDC will be the focus of the report: daily cases and death, vaccination and ICU admission rates, and vaccination rates for the countries of the EU/EEA.

To shed light on the COVID-19 situation in Europe right now, the study will examine and visualise the data. The analysis will be divided into sections that compare regional trends, ICU admission rates, trends in daily new cases and deaths, vaccination rates, and rates of vaccination and ICU admission. The use of materialised views in the report will also be investigated as a way to enhance the efficiency of frequently used queries.

Overall, by utilising ECDC datasets, this report seeks to provide a thorough understanding of the COVID-19 situation in Europe. To help public health officials and decision-makers in their efforts to stop the pandemic, they can use the knowledge gained from this analysis.

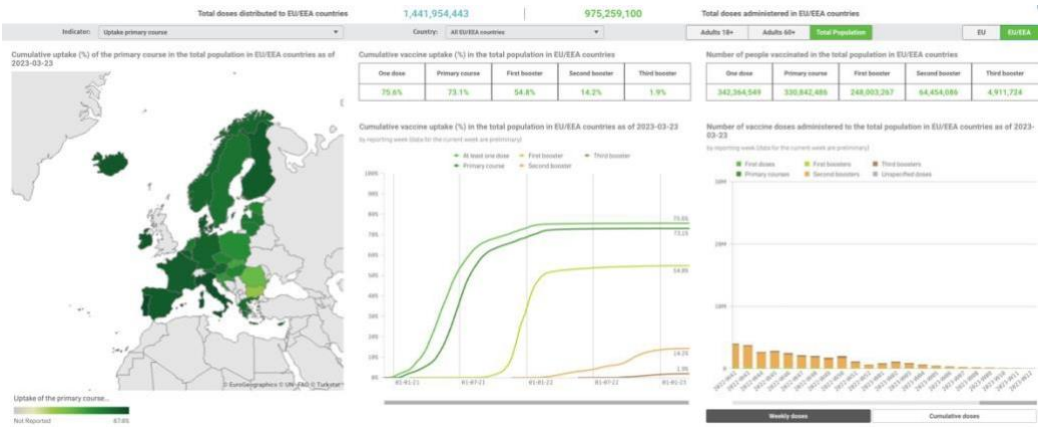



Figure 1: COVID-19 Vaccine Tracker dashboard

## A. Background information on the datasets

The information for this study was given by the European Centre for Disease Prevention and Control. Data on daily new cases o f COVID-19 in the EU/EEA, provides statistics on the number of recently reported cases and deaths by region, date, and location. Data from the EU/EEA The second dataset, "Data on hospital and ICU admission rates and current occupancy for COVID-19," offers specifics on vaccination and ICU admission rates as well as present patient occupancy for COVID-19 patients by nation

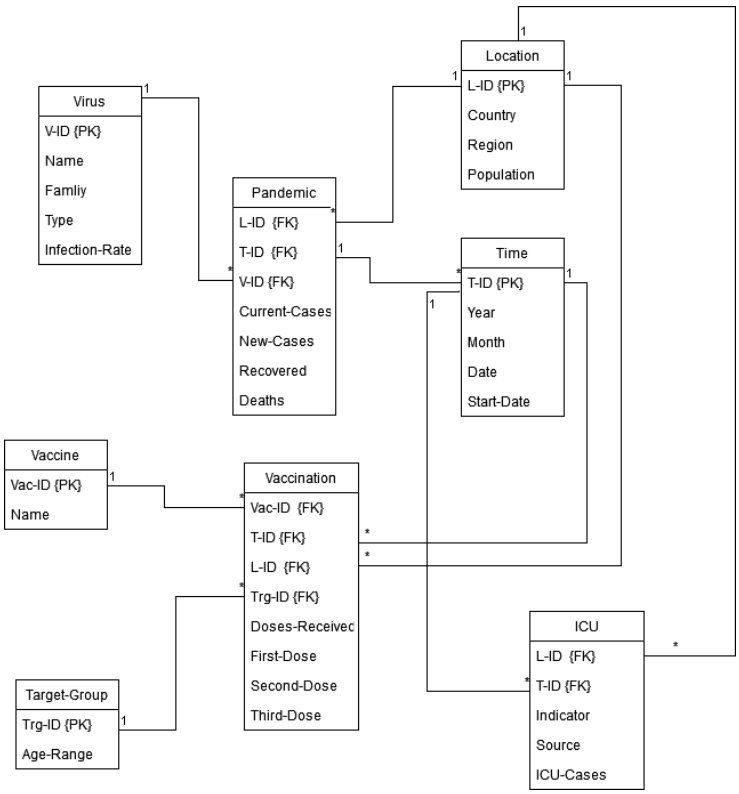
and time. These datasets are valuable resources for monitoring the impact of the COVID-19 pandemic and tracking the progress of vaccination efforts. The data is used to inform public health policies and decision-making at the national and international levels.

## B. Purpose of the report

The purpose of this report is to design and implement a data warehouse schema that can promptly answer the frequent queries '1-3' related to COVID-19 cases, deaths, ICU cases, and vaccination rates. The report includes the design of a data warehouse model (DFM), its mapping to a logical model, and its implementation using SQLITE and Python. Additionally, the report discusses the use of materialized views to improve the response time of frequent queries and provides the implementation of two materialized views to answer the directors' frequent queries '1-3' based on regions, months, and quarters. The report aims to provide a comprehensive solution to efficiently manage and analyse COVID-19 data to support decision-making processes.

## C. Scope and limitations of the report

The given relational database has three fact tables - Pandemic, Vaccination, and ICU - and five dimensions - Virus, Target Group, Time, Location, and Vaccine. The model is designed with functional dependencies to ensure structured data storage and prompt answers to frequent queries. The Pandemic table captures pandemic-related data, Vaccination records vaccine administration, and ICU stores ICU cases. The dimensions provide specific information about viruses, target groups, timeline, location, and vaccine types. This model allows easy aggregation and retrieval of data for frequent queries, enabling efficient data analysis.



## i. Functional Dependencies

**For Virus:**

V_ID -> Name

V_ID -> Family

V_ID -> Type

V_ID -> Infection-Rate

**For Pandemic:**

L_ID, T_ID, V_ID -> Current-Cases

L_ID, T_ID, V_ID -> New-Cases

L_ID, T_ID, V_ID -> Recovered

L_ID, T_ID, V_ID -> Deaths

**For Location:**

L_ID -> Country

L_ID -> Region

L_ID -> Population

**For Time:**

T_ID -> Year

T_ID -> Month

T_ID -> Date

T_ID -> Start-Date

**For Vaccine:**

Vac_ID -> Name

**For ICU:**L_ID, T_ID, Indicator, Source -> ICU-Cases

**For Vaccination:**

Vac_ID, T_ID, L_ID, Trg_ID -> Doses-Received

Vac_ID, T_ID, L_ID, Trg_ID -> First-Dose

Vac_ID, T_ID, L_ID, Trg_ID -> Second-Dose

Vac_ID, T_ID, L_ID, Trg_ID -> Third-Dose

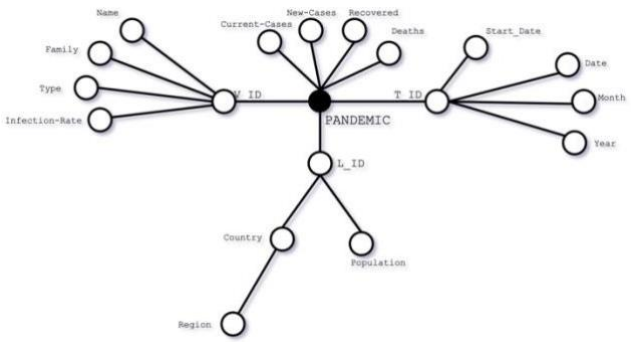
**For Target-Group:**

Trg_ID -> Age-Range

## ii. Build the Attribute Tree from the integrated relational schema

**The constructed attribute trees**

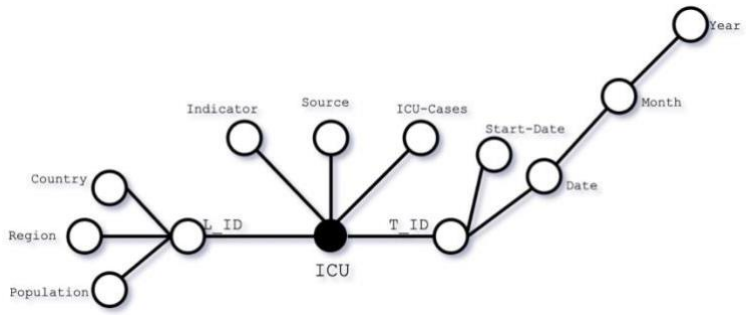
Pandemic Attribute Tree



The constructed attribute tree represents a part of the Pandemic data warehouse model. It contains three main branches: Virus ID, Location ID, and Time ID. The Virus ID entity has child attributes such as Name, Family, and Type, describing the virus's characteristics. The Location ID entity represents the country and region where the pandemic is spreading, along with population information. The Time ID entity provides the timeline of the pandemic. The tree also includes measures such as Current Cases, New Cases, Recovered, and Deaths, which provide information about pandemic-related data.

---

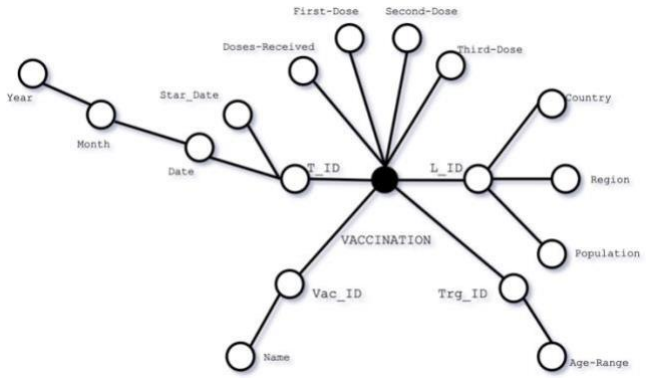
ICU attribute tree




The given attribute tree represents the hierarchical structure of the ICU fact table, with Location and Time being the primary dimensions. The fact table captures data related to ICU cases and includes attributes such as Indicator, Source, and ICU-cases.

---

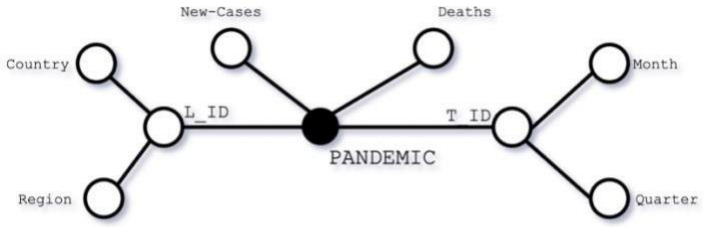
Vaccination Attribute Tree



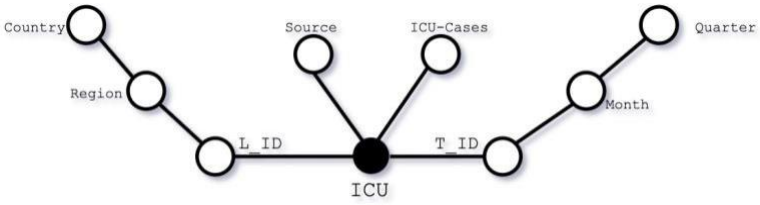

**Pruned Attribute Trees**

Pruning attribute trees involves removing unnecessary or redundant attributes in the process of designing data marts. This helps to simplify the overall design and improve query performance, as it reduces the amount of data that needs to be processed. By removing attributes that are not needed for answering specific business questions, the data mart becomes more focused and efficient in delivering the required information.

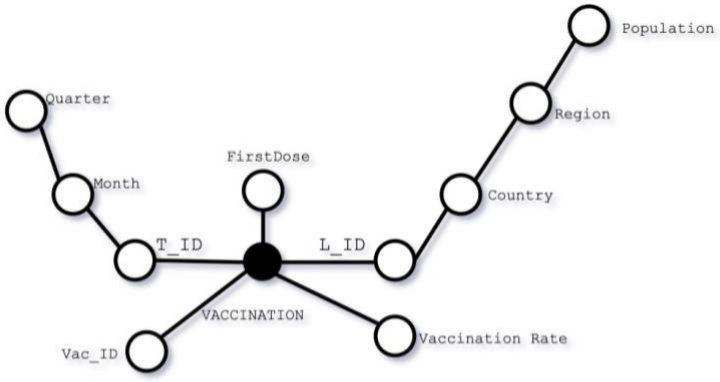
Pandemic Attribute Tree



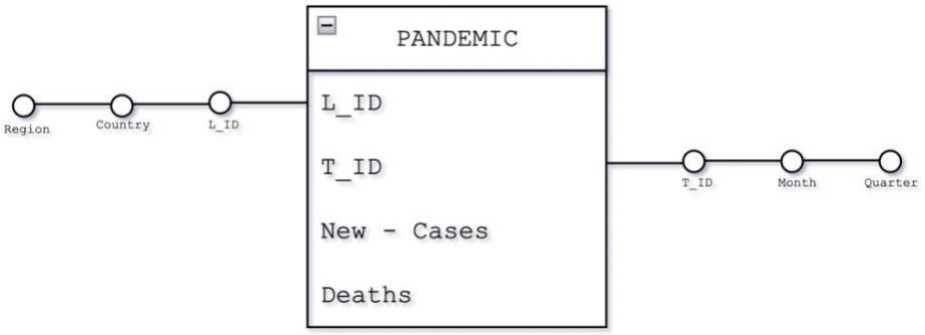

ICU Attribute Tree

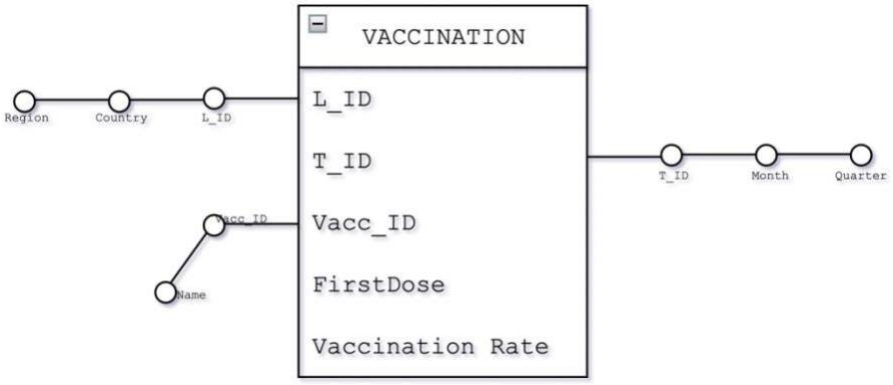


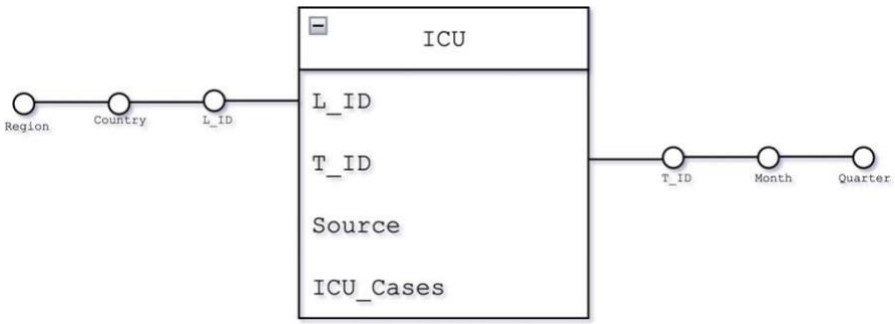

Vaccination Attribute Tree



- **Fact Schemas for each attribute tree:**



## iii. Build the Fact Schema (DFM) from Attribute Tree.

Covid-19 Data Analysis is the main fact table with has all the information needed in order to solve the frequent queries. The dimensions are the tables: Location, Vaccine and Time.

**Business Process**: Covid-19 Data Analysis

**Dimensions:**

1. Location
2. Time
3. Vaccine

**Measures**: cases, deaths, ICU_cases and vaccinationRate.

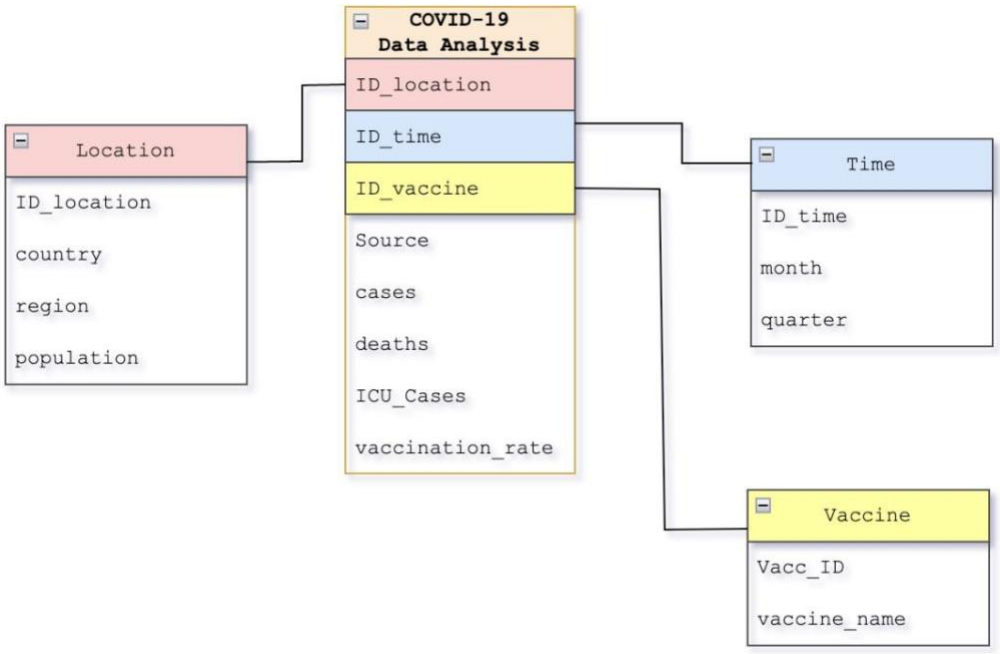
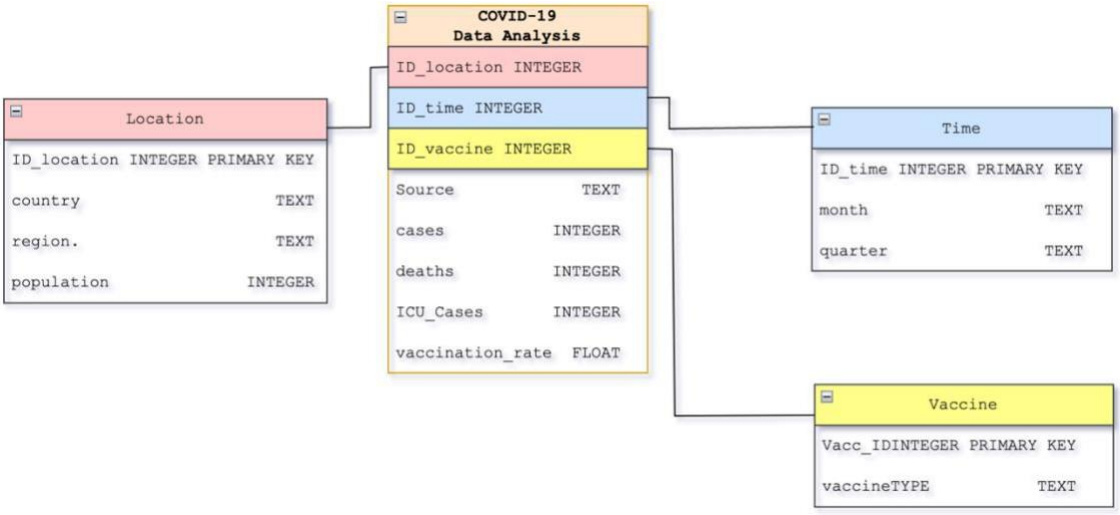



Figure 2: Dimensional Fact Model

**2. Map the DFM model to a logical model (i.e. relational). Clearly display the main fact table(s) and dimensions.**



### 3. Implement the above logic as a working data warehouse schema, under MySQL/SQLITE, or any other suitable DBMS. Provide the DDL statements to create the proposed data warehouse schema.

The code creates four tables: CovidData, TIME, LOCATION, and VACCINE. The CovidData table is the main fact table that contains information about COVID-19 cases and deaths, vaccination rates, and ICU cases in different countries and regions. The TIME table contains information about months and quarters, the LOCATION table contains information about countries and regions, and the VACCINE table contains information about vaccine types.

DDL Statements

| Table Name | DDL statement |
|---|---|
| **TIME Dimension** | `conn.execute('''CREATE TABLE TIME (`<br>`ID_time INT NOT NULL,`<br>`month TEXT,`<br>`quarter TEXT,`<br>`PRIMARY KEY (ID_time)`<br>`)''')` |
| **LOCATION Dimension** | `conn.execute(''' CREATE TABLE LOCATION (`<br>`ID_location INT NOT NULL,`<br>`country TEXT,`<br>`region TEXT,`<br>`population INTEGER,`<br>`Primary Key (ID_location)`<br>`)''')` |
| **VACCINE Dimension** | `conn.execute(''' CREATE TABLE VACCINE (`<br>`ID_vaccine INT NOT NULL,`<br>`vaccineType TEXT,`<br>`Primary Key (ID_vaccine)`<br>`)''')` |
| **CovidData Business Processes** | `conn.execute(`<br>`'''CREATE TABLE CovidData`<br>`(ID_time INTEGER NOT NULL,`<br>`ID_location INTEGER NOT NULL,`<br>`ID_vaccine INTEGER NOT NULL,`<br>`cases INTEGER,`<br>`deaths INTEGER,`<br>`firstDose INTEGER,`<br>`population INTEGER,`<br>`vaccinationRate REAL,`<br>`ICU_Cases INTEGER,`<br>`source TEXT,`<br>`PRIMARY KEY (ID_time, ID_location, ID_vaccine),`<br>`FOREIGN KEY (ID_time) REFERENCES TIME(ID_time),`<br>`FOREIGN KEY (ID_location) REFERENCES LOCATION(ID_location),`<br>`FOREIGN KEY (ID_vaccine) REFERENCES VACCINE(ID_vaccine));''')` |

## 4. Considering the designed data warehouse and its cardinalities, decide whether and which materialized views are convenient to improve the response time of the frequent queries (consider all the frequent queries). Explain the reasons for your choices

Based on the designed data warehouse, there are several frequent queries that can benefit from materialized views to improve response time.

One of the most frequent queries is the one that reports cases and deaths for each region and month. Since this query involves joining the CovidData, location, and time tables, a materialized view that stores the results of this query can significantly improve response time. Therefore, we can create a materialized view named covid_summary with the following query:

```
# create materialized view for covid summary
conn.execute("CREATE MATERIALIZED VIEW covid_summary AS "
             "SELECT location.region, time.month, SUM(cases) AS total_cases, SUM(deaths) AS total_deaths "
             "FROM CovidData "
             "JOIN location ON CovidData.ID_location = location.ID_location "
             "JOIN time ON CovidData.ID_time = time.ID_time "
             "GROUP BY location.region, time.month "
             "ORDER BY location.region, time.month ")

```

The query that reports the total number of cases and deaths for each nation and vaccine type is another common one that could use materialised view. The CovidData, location, and vaccine tables are joined in this query. However, the resulting materialised view might be too big to be useful because the CovidData table has a cardinality of roughly 10,000 records. As a result, we might think about developing an indexed view instead, which would be better for this query.

## 5. Provide and implement a materialised view(s) to answer the director's frequent queries '1-3'

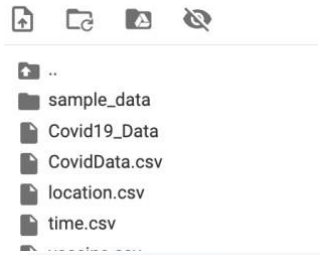
Firstly I imported the required libraries, including sqlite3 for interacting with SQLite databases and pandas for working with data frames.

```
import sqlite3
import pandas as pd
```

Then it is the time to read four CSV files, CovidData.csv, location.csv, time.csv, and vaccine.csv, into four separate data frames using the pd.read_csv() function.

```
df_CovidData_ = pd.read_csv('CovidData.csv')
df_location = pd.read_csv('location.csv')
df_time = pd.read_csv('time.csv')
df_vaccine = pd.read_csv('vaccine.csv')
```

Then I createda new SQLite database called "Covid19_Data" using the sqlite3.connect() function and assigns it to the variable conn.The code creates a new cursor object using the conn.cursor() function and assigns it to the variable cur. The cursor object is used to execute SQL statements and interact with the database.




```
# connect to database
conn = sqlite3.connect("Covid19_Data")
cur = conn.cursor()
```

```
df_CovidData_
```

| | ID_time | ID_vaccine | ID_location | cases | deaths | firstDose | population | vaccinationRate | ICU_Cases | source |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 3 | 1 | 877 | 3557.0 | 0.0 | 3.0 | 8901064 | 0.00% | 856.0 | Twitter |
| 1 | 3 | 1 | 878 | 5494.0 | 4.0 | 0.0 | 8901064 | 0.00% | 856.0 | Twitter |
| 2 | 3 | 1 | 879 | 7776.0 | 4.0 | 0.0 | 8901064 | 0.00% | 856.0 | Twitter |
| 3 | 3 | 1 | 880 | 8221.0 | 6.0 | 0.0 | 8901064 | 0.00% | 856.0 | Twitter |
| 4 | 3 | 1 | 881 | 10007.0 | 8.0 | 0.0 | 8901064 | 0.00% | 856.0 | Twitter |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 7803 | 30 | 5 | 910 | NaN | NaN | NaN | 50400 | 0.00% | NaN | NaN |
| 7804 | 30 | 5 | 911 | NaN | NaN | NaN | 50400 | 0.00% | NaN | NaN |
| 7805 | 30 | 5 | 912 | NaN | NaN | NaN | 50400 | 0.00% | NaN | NaN |
| 7806 | 30 | 5 | 913 | NaN | NaN | NaN | 50400 | 0.00% | NaN | NaN |
| 7807 | 30 | 5 | 914 | NaN | NaN | NaN | 50400 | 0.00% | NaN | NaN |

7808 rows × 10 columns

```
# load dataframes into database as tables with names CovidData, TIME, LOCATION, VACCINE
df_CovidData_.to_sql("CovidData", conn)
df_time.to_sql('TIME',conn)
df_vaccine.to_sql('VACCINE',conn)
df_location.to_sql('LOCATION',conn)
```

**Then I used this Python code to insert data into the database table called "Covid19_Data". I repeated the same step for the other 3 dimension tables.**

```
for i in range(1, 100):
    conn.execute("INSERT INTO Covid19_Data
(ID_time,ID_vaccine,ID_location,cases,deaths,firstDose,population,vaccinationRate,ICU_Cases,so
urce) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)", (i, i*2, i*4, i*8, i*16, i*32, i*64, i*128,
i*256,i*512))
    conn.commit()
```

**Overall these are the steps I followed:**

- Import necessary Python libraries: sqlite3, pandas, and csv.
- Connect to the SQLite database and create a cursor object.
- Read CSV files containing data for time, location, vaccine, and CovidData, and load them into corresponding tables in the database.
- Define the schema for the dimension tables time, location, and vaccine, and create them in the database using the SQL CREATE TABLE statement.
- Define the schema for the main fact table CovidData, and create it in the database using the SQL CREATE TABLE statement.
- Insert data into CovidData, time, location, and vaccine tables using the SQL INSERT INTO statement.
- Create a materialized view named covid_summary that summarizes CovidData by region and month.
- Execute two SQL queries and store the results in Pandas DataFrames.
- Print the results

**For each region and month report:**

- the COVID-19 cases and deaths

```
# Define the SQL query for creating the view
# For each region and month report the COVID-19 cases and deaths

query1 = '''
CREATE MATERIALIZED VIEW covid_cases_and_deaths_by_region_and_month
AS
SELECT location.region, time.month,  SUM(cases) AS total_cases, SUM(deaths) AS total_deaths
FROM Covid19_Data
JOIN location ON Covid19_Data.ID_location = location.ID_location
JOIN time ON Covid19_Data.ID_time = time.ID_time
GROUP BY location.region, time.month
'''
cur = conn.cursor()
cur.execute(query1)
conn.commit()

query2 = '''
SELECT *
FROM covid_cases_and_deaths_by_region_and_month
ORDER BY region, month
'''

results = pd.read_sql(query2, conn)
print(results)
```

```
   region   month  total_cases  total_deaths
0     112  Aug-03         56.0         112.0
1     128  Aug-03         64.0         128.0
2     144  Aug-03         72.0         144.0
3      16  Jul-03          8.0          16.0
4     160  Aug-03         80.0         160.0
5     176  Oct-03         88.0         176.0
```

- **Intensive care unit (ICU)cases as reported by different sources**

```
# Define the SQL query for creating the materialized view
# Intensive care unit (ICU) cases as reported by different sources
query3 = '''
CREATE MATERIALIZED VIEW covid_icu_cases_by_region_and_month AS
SELECT location.region, time.month, Covid19_Data.source, SUM(ICU_Cases) AS total_ICU_Cases
FROM Covid19_Data
JOIN location ON Covid19_Data.ID_location = location.ID_location
JOIN time ON Covid19_Data.ID_time = time.ID_time
GROUP BY location.region, time.month, Covid19_Data.source
'''

cur.execute(query3)


cur.execute('REFRESH MATERIALIZED VIEW covid_icu_cases_by_region_and_month')

query4 = '''
SELECT *
FROM covid_icu_cases_by_region_and_month
ORDER BY source, region, month
'''


results = pd.read_sql(query4, conn)

print(results)
```

```
   region   month source  total_ICU_Cases
0      32  Jul-03   1024            512.0
```

- **vaccination rates for each vaccine.**

```
# Define the SQL query for creating the materialized view
# For each region and month report the vaccine rates by each vaccine
query = '''
CREATE MATERIALIZED VIEW IF NOT EXISTS covid_vaccine_rate_by_region_and_month_mv
AS
SELECT location.region, time.month, vaccine.vaccineType, Covid19_Data.vaccinationRate
FROM Covid19_Data
JOIN location ON Covid19_Data.ID_location = location.ID_location
JOIN time ON Covid19_Data.ID_time = time.ID_time
JOIN vaccine ON Covid19_Data.ID_vaccine = vaccine.ID_vaccine
GROUP BY location.region, time.month, vaccine.vaccineType
ORDER BY vaccine.vaccineType
'''

cur.execute(query)
conn.commit()

query = '''
SELECT *
FROM covid_vaccine_rate_by_region_and_month_mv
'''
results = pd.read_sql_query(query, conn)
print(results)
```

| | region | month | vaccineType | vaccinationRate |
|---|---|---|---|---|
| 0 | 48 | Jul-03 | 12 | 384 |
| 1 | 64 | Jul-03 | 16 | 512 |
| 2 | 80 | Jul-03 | 20 | 640 |
| 3 | 96 | Aug-03 | 24 | 768 |

**For each country and quarter report:**

- the COVID-19 cases and deaths,

```
# for each country and quarter show the covid cases and deaths
query = '''
CREATE MATERIALIZED VIEW covid_cases_and_deaths_by_country_and_quarter
AS
SELECT location.country, time.quarter, SUM(cases) AS total_cases, SUM(deaths) AS total_deaths
FROM Covid19_Data
JOIN location ON Covid19_Data.ID_location = location.ID_location
JOIN time ON Covid19_Data.ID_time = time.ID_time
GROUP BY location.country, time.quarter
ORDER BY location.country, time.quarter
'''

cur.execute(query)

conn.commit()

query = '''
SELECT *
FROM covid_cases_and_deaths_by_country_and_quarter
'''

results = pd.read_sql(query, conn)

print(results)
```

- Intensive care unit cases

```
# for each country and quarter show the icu cases
query = '''
CREATE MATERIALIZED VIEW covid_icu_cases_by_country_and_quarter_mview AS
SELECT location.country, time.quarter, Covid19_Data.source, SUM(ICU_Cases) AS total_ICU_Cases
FROM Covid19_Data
JOIN location ON Covid19_Data.ID_location = location.ID_location
JOIN time ON Covid19_Data.ID_time = time.ID_time
GROUP BY location.country, time.quarter
ORDER BY Covid19_Data.source
'''

cur.execute(query)

conn.commit()

```

- vaccination rates for each vaccine.

```
# for each country and quarter show the vaccination rates
query = '''
CREATE MATERIALIZED VIEW vaccination_rates_by_country_and_quarter_matview AS
SELECT location.country, time.quarter, vaccine.vaccineType, Covid19_Data.vaccinationRate
FROM Covid19_Data
JOIN location ON Covid19_Data.ID_location = location.ID_location
JOIN time ON Covid19_Data.ID_time = time.ID_time
JOIN vaccine ON Covid19_Data.ID_vaccine = vaccine.ID_vaccine
GROUP BY location.country, time.quarter, vaccine.vaccineType
ORDER BY vaccine.vaccineType
'''

cur.execute(query)

conn.commit()
```

VISUALISATION OF THE DATA for query 1

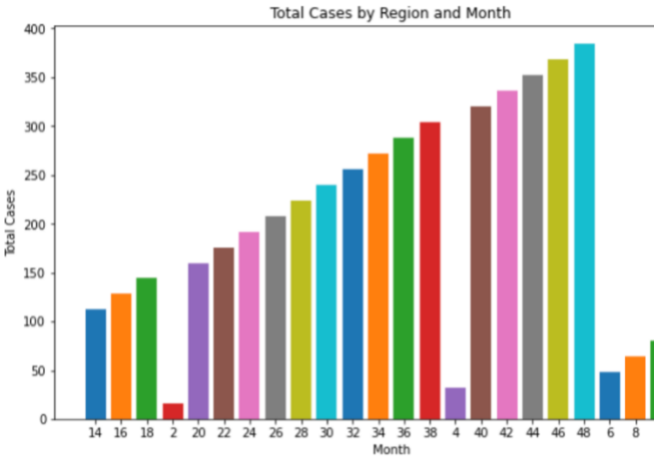
```python
import matplotlib.pyplot as plt

query1 = '''
SELECT location.region, time.month, SUM(cases) AS total_cases, SUM(deaths) AS total_deaths
FROM Covid19_Data
JOIN location ON Covid19_Data.ID_location = location.ID_location
JOIN time ON Covid19_Data.ID_time = time.ID_time
GROUP BY location.region, time.month
ORDER BY location.region, time.month
'''

results = pd.read_sql(query1, conn)

fig, ax = plt.subplots(figsize=(10, 6))
for i, group in results.groupby('region'):
    ax.bar(group['month'], group['total_cases'], label=i)
ax.set_xlabel('Month')
ax.set_ylabel('Total Cases')
ax.set_title('Total Cases by Region and Month')
ax.legend()
plt.show()


```




## D. CONCLUSION

In summary, designing a data mart requires a number of steps that are essential for its implementation. To make sure that the data mart satisfies the demands of the end users, the process begins with an analysis of the data sources and knowledge of the requirements. The structure of the data mart and the connections between the different data items are defined during the conceptual and logical design phases. The data mart can handle the anticipated query volumes with the help of workload analysis and refining, which also helps to maximise performance. For the data mart to be implemented and made accessible to end users, the data staging and physical design phases are essential. Overall, if these steps are understood and followed, the data mart may be created in a way that is both effective and efficient for the organisation's needs.