Covid Data Analysis in SQL

For this exercise I used mySQL to analyze Covid data. The data is downloaded from "Our World in Data" website (https://ourworldindata.org/covid-deaths). The tools used are mySQL and DBeaver as database administration tool.

DBeaver is a free database administration tool that supports multiple variants of SQL. It can be downloaded at https://dbeaver.io/download/ .

I intended to use MySQL Workbench for this project, however there was a bug related to uploading data from csv format into MySQL Workbench which prevented me from having data uploaded correctly. The Workbench ran into an issue uploading data that had null values and missing values. Additionally, it took over 2 hours to upload a 40MB file.

Here is the link to MySQL documentation reference manual https://dev.mysql.com/doc/refman/8.3/en/sql-data-definition-statements.html

The goal of this exercise is to review and practice the use of Joins, CTE's, Temp Tables, Window Functions, Aggregate Functions, Creating Views, Converting data types. Before going to SQL code let's review these concepts.

Joins

SQL Join operation combines data from two or more tables based on a common column (one or more) between them. There are four most used types of Joins in SQL to consider Inner Join, Outer Join, Left join, and Right Join.

Inner Join – Select rows from A where there is a match from B.

Left Join – Select all rows from A plus rows where there is a match in B.

Right Join – Select all rows in B plus rows where there is a match in A.

Full Outer Join – Select all rows from A and all rows from B even where there is no match.

Aggregate Functions

An aggregate function is a function that performs a calculation on a set of values and returns a single value. Aggregate functions are often used with the GROUP BY clause of the SELECT statement. The GROUP BY clause splits the result-set into groups of values and the aggregate function can be used to return a single value for each group.

The most commonly used SQL aggregate functions are:

MIN() - returns the smallest value within the selected column

MAX() - returns the largest value within the selected column

COUNT() - returns the number of rows in a set

SUM() - returns the total sum of a numerical column

AVG() - returns the average value of a numerical column

Aggregate functions ignore null values (except for COUNT()).

Window Functions

A window function performs a calculation across a set of table rows (a window) that are somehow related to the current row. This is comparable to the type of calculation that can be done with an aggregate function. But unlike regular aggregate functions, use of a window function does not cause rows to become grouped into a single output row — the rows retain their separate identities.

There are three different types of window functions: aggregate window functions, value window functions, and ranking window functions.

Aggregate window functions are used to perform operations on sets of rows in a window(s). They include SUM(), MAX(), COUNT(), and others.

Value window functions are like aggregate window functions that perform multiple operations in a window, but they're different from aggregate functions. They include things like LAG(), LEAD(), FIRST_VALUE(), and others. We will see their usefulness later in the section.

Rank window functions are used to rank rows in a window(s). They include RANK(), DENSE_RANK(), ROW_NUMBER(), and others.

OVER clause is used with window functions to define that window. OVER clause does two things:

Partitions rows to form a set of rows. (PARTITION BY clause is used)

Orders rows within those partitions into a particular order. (ORDER BY clause is used)

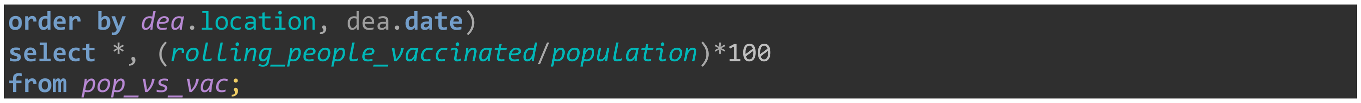
CTE's

CTE is short for common table expressions. CTEs work as virtual tables created during the execution of a query, used by the query, and eliminated after query execution. CTEs make a complex query easier to read.

A CTE is a named subquery that appears on the top of a query in a WITH clause before SELECT, INSERT, UPDATE, DELETE, or MERGE statement. The WITH clause can include one or more CTEs separated by commas.

```sql
-- Using CTE to perform calculation on partition by in previous query

with pop_vs_vac (continent, location, date, population, new_vaccinations,
rolling_people_vaccinated)
as (
select dea.continent, dea.location, dea.date, dea.population, vac.new_vaccinations,
SUM(vac.new_vaccinations) over (
partition by dea.location
order by dea.location, dea.date) as rolling_people_vaccinated
from covid_deaths dea
join covid_vaccinations vac
on dea.location = vac.location
and dea.date = vac.date
where dea.continent !=''
```



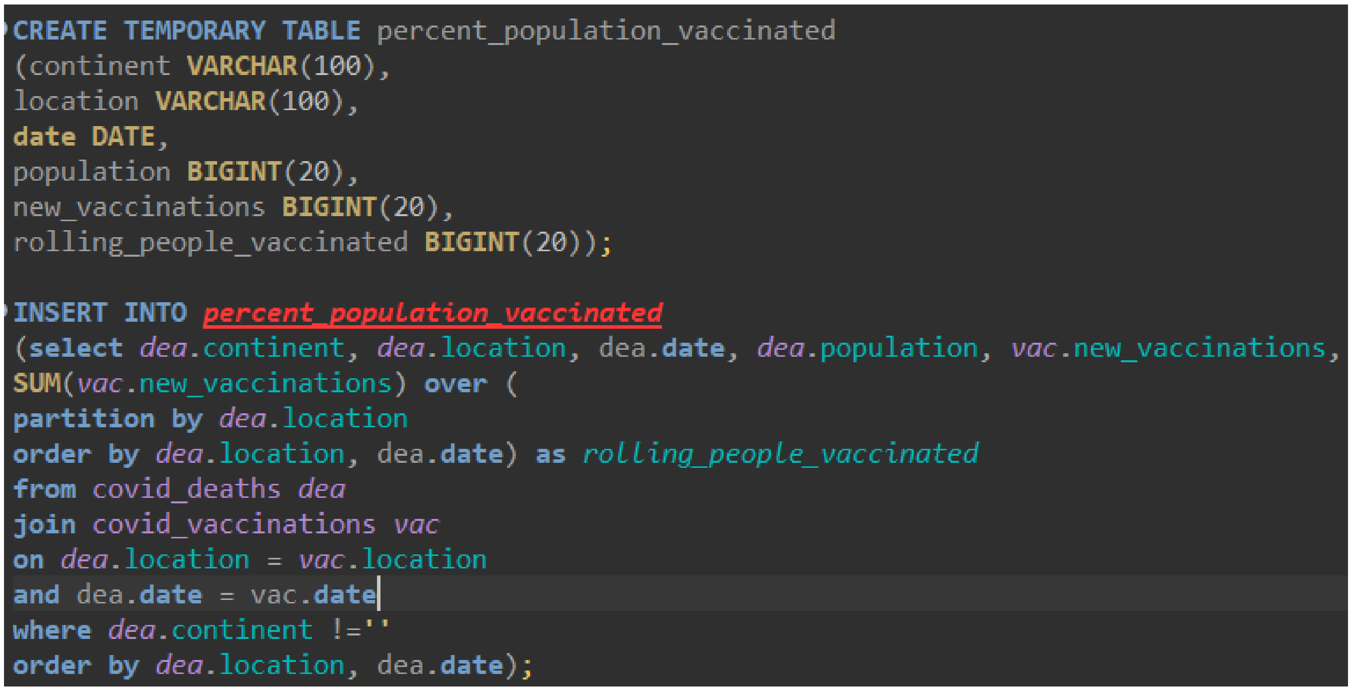
```
order by dea.location, dea.date)
select *, (rolling_people_vaccinated/population)*100
from pop_vs_vac;
```

Temp Tables

A temporary table is a table that is created and used within the context of a specific session in a database management system. It is designed to store temporary data that is needed for a short duration and does not require a permanent storage solution. Temporary tables are dropped when the session that creates the table has closed or can also be explicitly dropped by users.

Temporary tables are created on the fly and are typically used to perform complex calculations, store intermediate results, or manipulate subsets of data during the execution of a query or a series of queries. Temporary tables in SQL provide a convenient way to break down complex problems into smaller, more manageable steps.

To create a temp table use CREATE TEMPORARY TABLE statement before the table name.



```
CREATE TEMPORARY TABLE percent_population_vaccinated
(continent VARCHAR(100),
location VARCHAR(100),
date DATE,
population BIGINT(20),
new_vaccinations BIGINT(20),
rolling_people_vaccinated BIGINT(20));

INSERT INTO percent_population_vaccinated
(select dea.continent, dea.location, dea.date, dea.population, vac.new_vaccinations,
SUM(vac.new_vaccinations) over (
partition by dea.location
order by dea.location, dea.date) as rolling_people_vaccinated
from covid_deaths dea
join covid_vaccinations vac
on dea.location = vac.location
and dea.date = vac.date
where dea.continent !=''
order by dea.location, dea.date);
```

Figure 1 Create and Populate Temp Table

View

A view is a virtual table that can be treated like a real table. However, views do not hold the actual data. It is useful when you need to refer to a query repeatedly, you can for example create a view where you join three tables together and then refer to this new resulting table by the view name to run it instead of rewriting the code for joining the tables every time. A view is a virtual table environment that's created from one or more tables to make it easier to work with data. You can think of a view as a query for which you have SELECT statement and given it a name.

A view will appear in your table structure once you run the code.

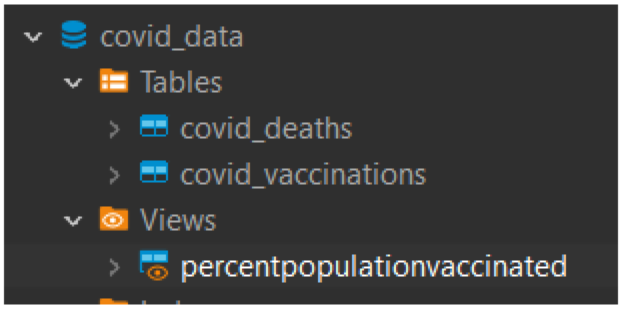
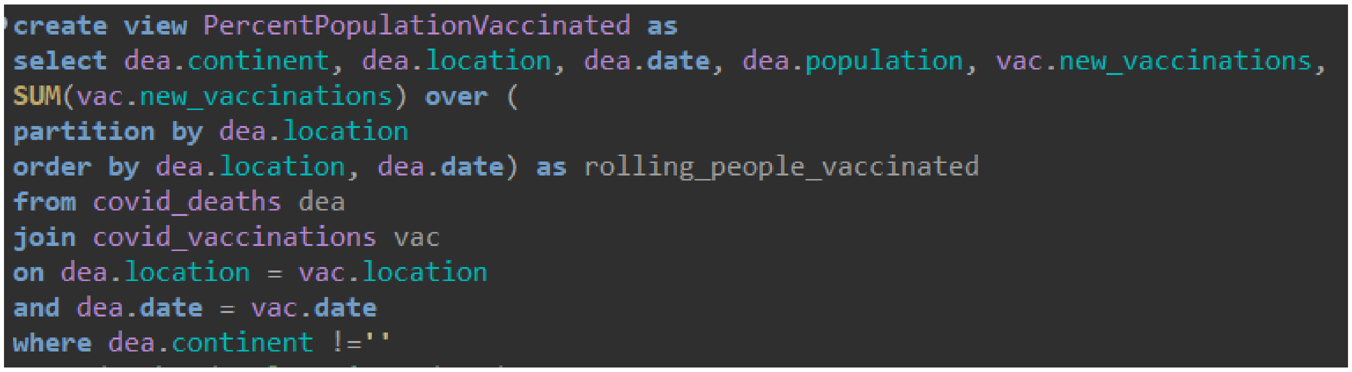


Figure 2 View in the Database Structure



```
create view PercentPopulationVaccinated as
select dea.continent, dea.location, dea.date, dea.population, vac.new_vaccinations,
SUM(vac.new_vaccinations) over (
partition by dea.location
order by dea.location, dea.date) as rolling_people_vaccinated
from covid_deaths dea
join covid_vaccinations vac
on dea.location = vac.location
and dea.date = vac.date
where dea.continent !=''
```

Figure 3 Create View

Converting data types

To convert a value from one data type to another use CONVERT() function or CAST() function.

For example:

SELECT CAST(column_name AS INTEGER)

FROM table_name;

Or

SELECT CONVERT(column_name, INTEGER)

FROM table_name;