How to use SQL with large Datasets

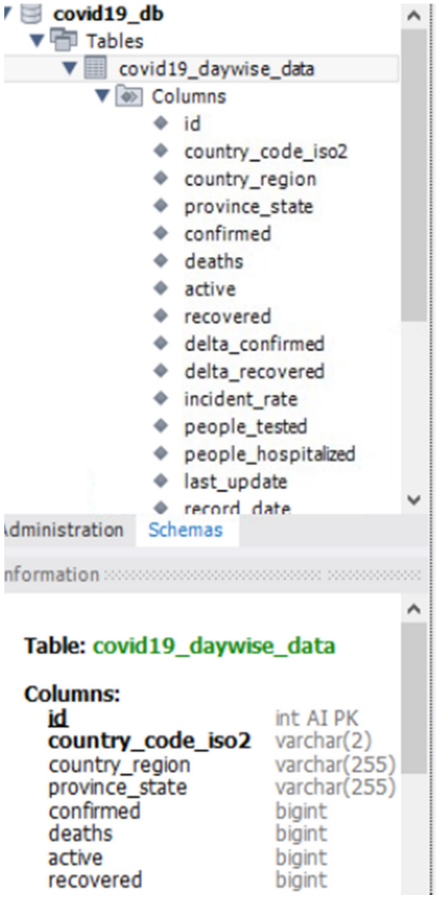
For this project we have used MySQL Workbench via cloud desktop.

If interested on doing it yourself please visit: https://www.coursera.org/projects/how-use-sql-large-datasets

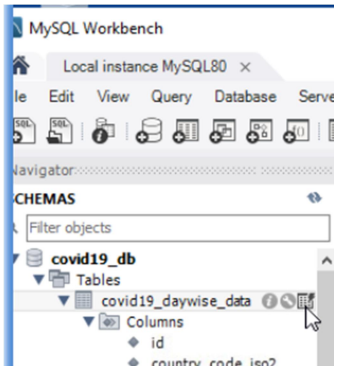
The dataset used can be found here: https://github.com/mahabub81/covid-19-api/blob/master/covid-19-mysql-dump/covid19_daywise_data.sql

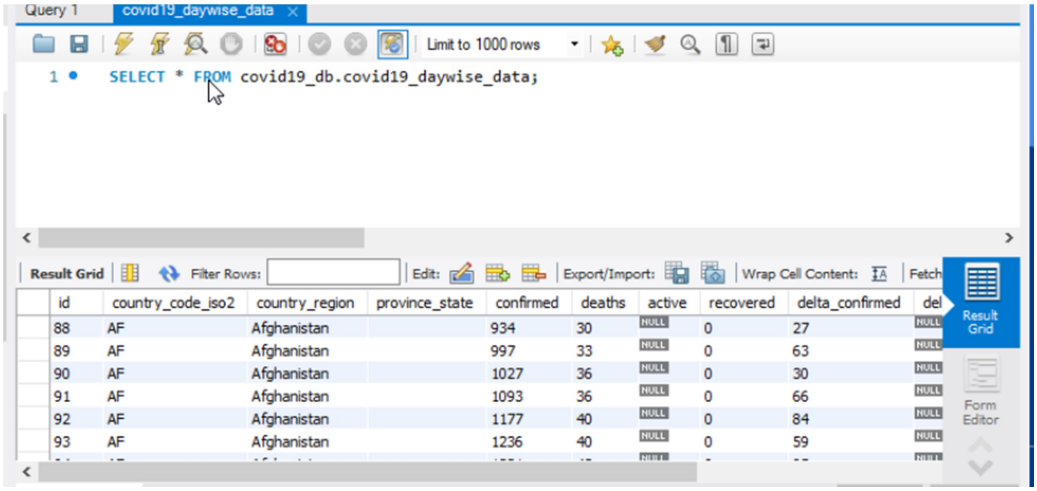
Task 1: Analyse an existing dataset

After connecting to the Local Instance, on the left Panel “Schemas” we can navigate the data base and see what columns contain. Also at the same time in the left bottom panel we can explore what data type is each:



We can execute a query by selecting the box that appears when hovering over the data set, which generates a basic SELECT * FROM query for us:



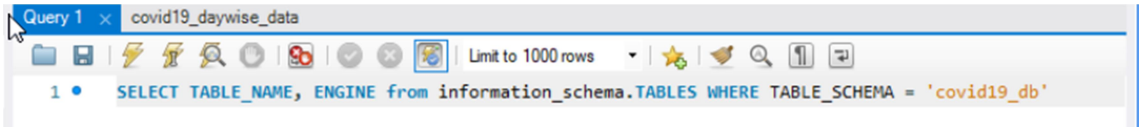


Is important to know the cases in this data base are cumulative over time. Knowing this will detract us from summing columns avoiding incorrect results

What kind of data storage engine is being used?

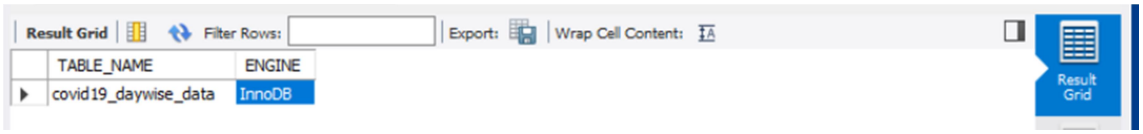
We can create a new empty blank query by clicking on the following symbol: 

This will be our Query 1:



We run it by clicking on the following symbol: 

On the middle panel we get our result grid, here we will see the Engine:



This is the default engine for MySQL now and needs to be invoked when we create the table.

If we wanted to recreate the table, we can change the schema, but for this exercises we will stick with InnoDB.

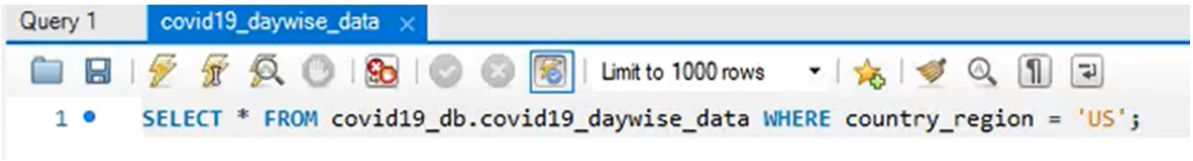
This is better for large data sets where a lot of concurrent reads may occur and also is being updated overtime.

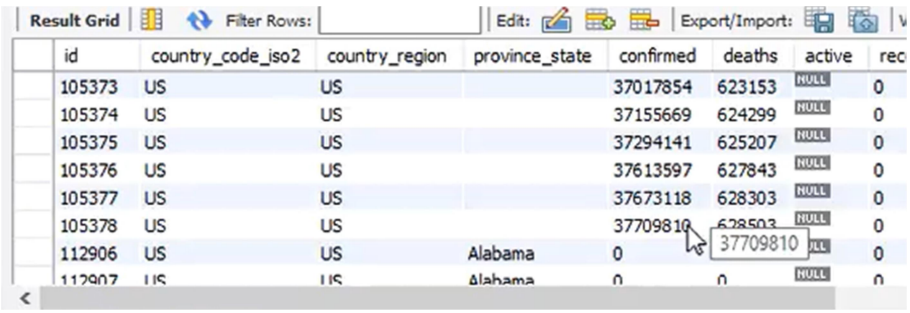
Task 2: Create queries to verify data values

a. Create a query to select US totals

In this case scenario we want to check the statistics only for US.

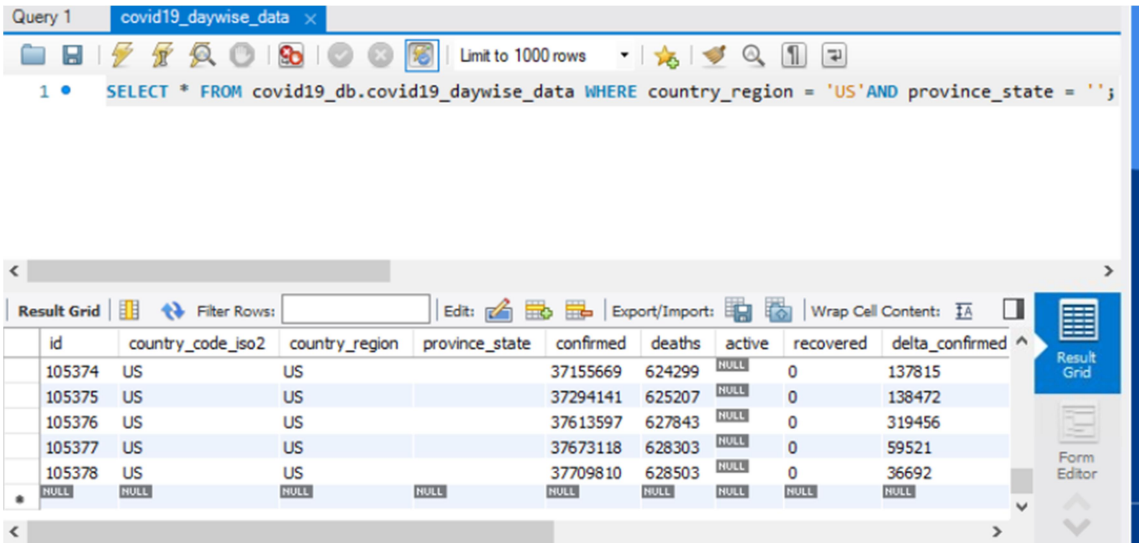
We can use the initial SELECT * FROM query and build it up to:





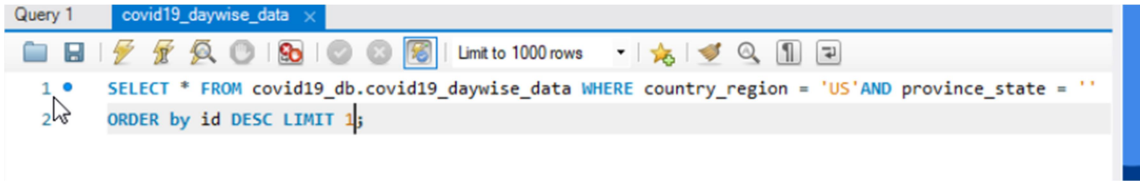
By the results, we know they accumulate totals for the entire US because there is a whole set of empty province_state.

So let's get the US totals where the province state is empty, as for the moment, we don't want it by province but totals:

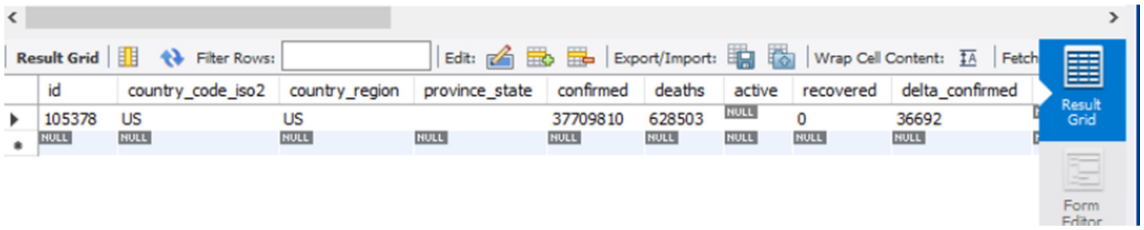


But precisely because we want the total and we know this data is cumulative, why not getting just the very last one?

For this we will use ORDER BY de id (as these are consecutive) in a DESC order (largest to smallest) and limit it to 1 so we get that entry alone:



And this is our result:



a. Check if sum of state totals match US totals and Check for valid state names

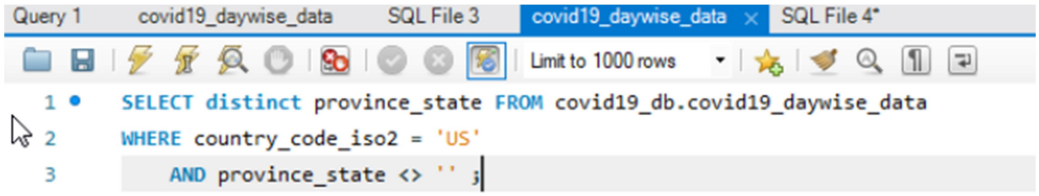
When analysing data is important to verify that what we are looking at is actually correct.

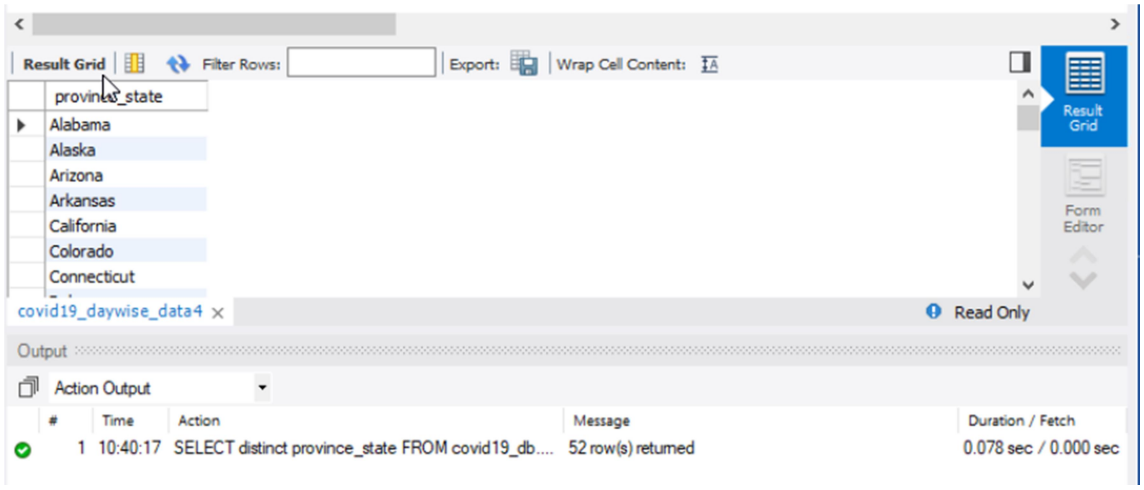
The approach will be verifying the Province States count

For this we use DISTINCT to get unique values (avoiding duplications)

Also let's use country_code_iso2 instead (we will explain later why)

Also, we don't want those in which province state is an empty value (because we already know that those are the cumulative totals for US)





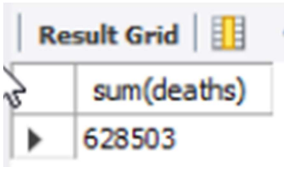
Now we get each province state in the result grid, and in the action output the total of rows returned, 52, sounds about right, all states accounted for!

Let's do a new query to check if the state totals for deaths matches the total in the database

By selecting MAX Record_date we select the last date on the table (remember, data is cumulative):

```
SELECT sum(deaths) FROM covid19_daywise_data
WHERE province_state is not NULL
    AND province_state <> ''
    AND country_region = 'US'
    AND record_date = (SELECT MAX(record_date) FROM covid19_daywise_data);
```

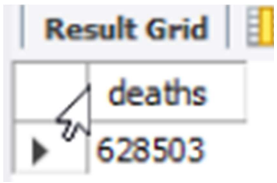
Our result:



Now let's verify it:

```
6    SELECT deaths FROM covid19_daywise_data
7    WHERE (province_state IS NULL OR province_state = '')
8        AND country_region = 'US'
9        AND record_date = (SELECT MAX(record_date) FROM covid19_daywise_data) ;
```

And we indeed get the same total:



That verifies that the data we're seeing is valid and matches what we see in the individual state totals

We could use the same approach to verify other data such as Confirmed cases

Task 3: Create a SQL index to improve query performance

On large datasets indexing can make all the difference.

One example of an index is the primary key which uniquely identifies a table row in the database, though we can also use indexing on other columns.

For example, when a query contains a date field in a WHERE clause, rather than looking for a given date in every single row and then obtaining those, the index has a reference to each row contains the date which can vastly improve performance to illustrate this.

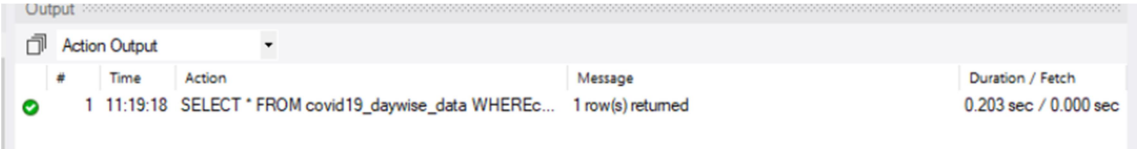
```
1 •   SELECT * FROM covid19_daywise_data
2     WHERE   country_region = 'US'
3         AND (province_state IS NULL OR province_state = '')
4         ORDER BY last_update DESC LIMIT 1;
```

It's great when tables also contain non unique values such as dates

For example, when a query contains a date field in a WHERE clause, rather than looking for a given date in every single row and then obtaining those, the index has a reference to each row contains the date which can vastly improve performance to illustrate this.

So let's go ahead and check the execution time in this query:

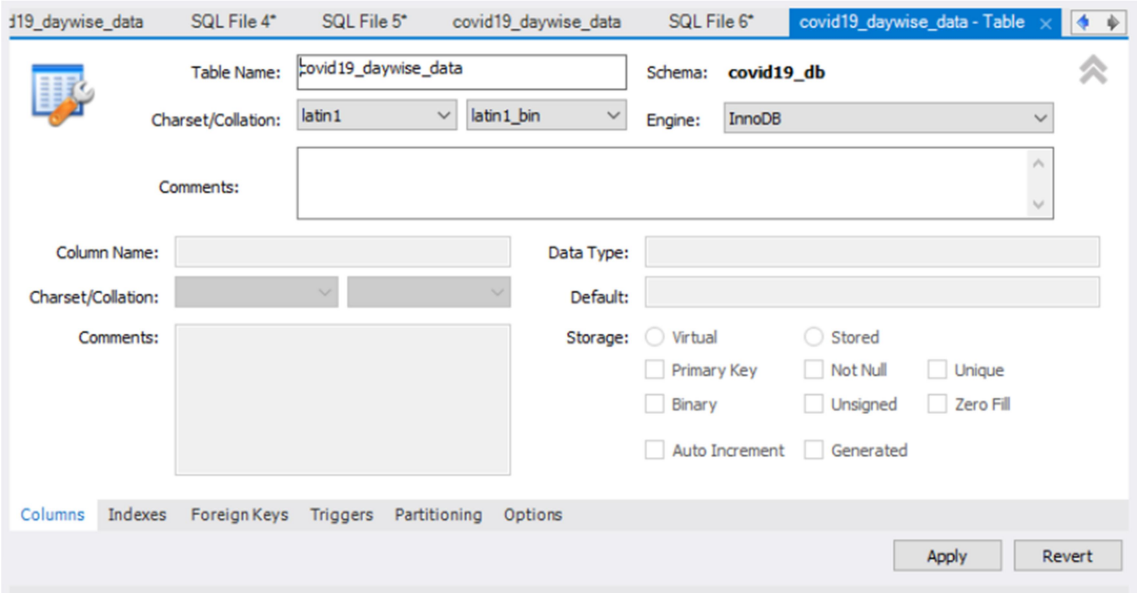
```
1 •   SELECT * FROM covid19_daywise_data
2     WHERE   country_region = 'US'
3         AND (province_state IS NULL OR province_state = '')
4         ORDER BY last_update DESC LIMIT 1;
```



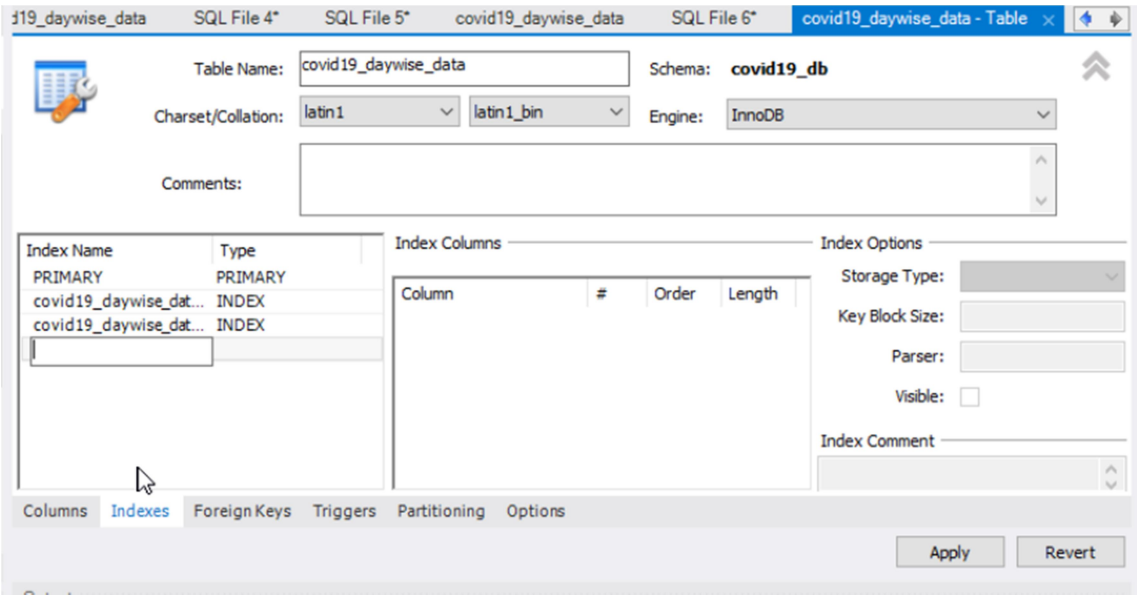
Let’s see what happens when we add an INDEX

For that we select the tool:

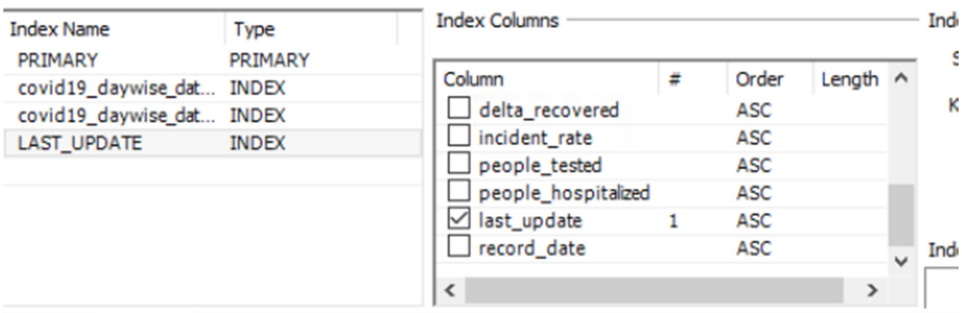
And we will see the following screen:



Then we go on to the tab Indexes (at the bottom)

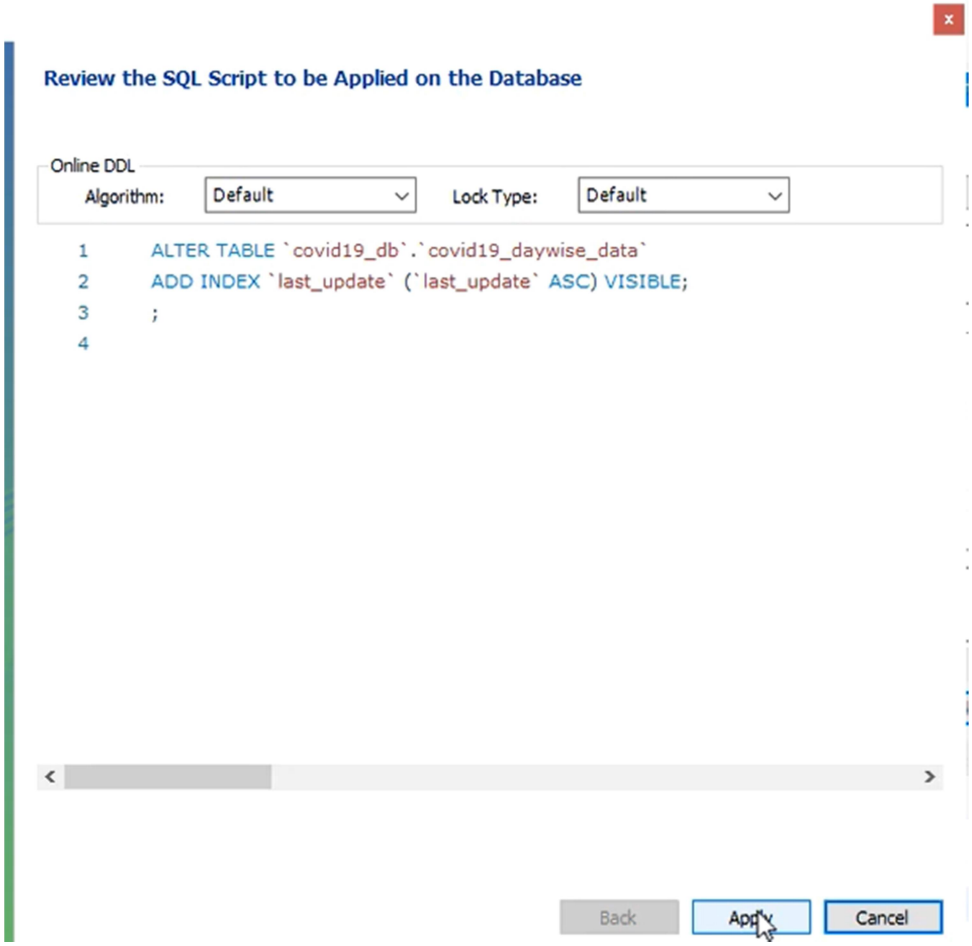


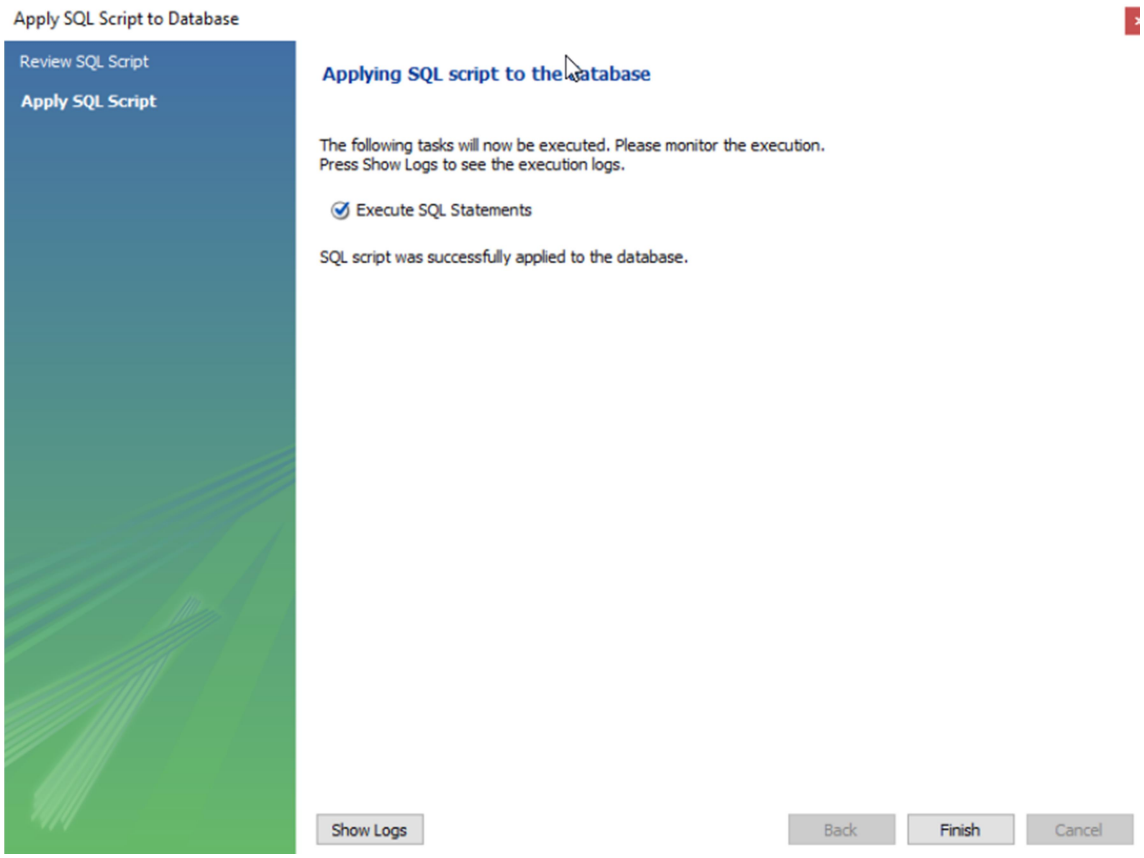
Let's create a new index by giving it a name and selecting Type INDEX in the left panel and then selecting last_update column in the Index Columns panel:



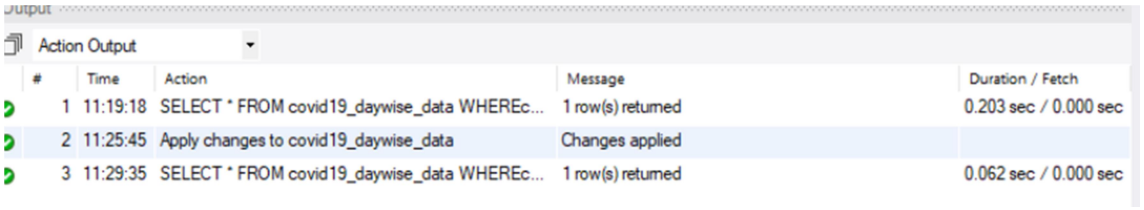
Then we click apply which brings us the SQL summary to create the LAST_UPDATE index.

In here we can review the query and modify if necessary, otherwise we will click apply again, and then finish one has executed it:

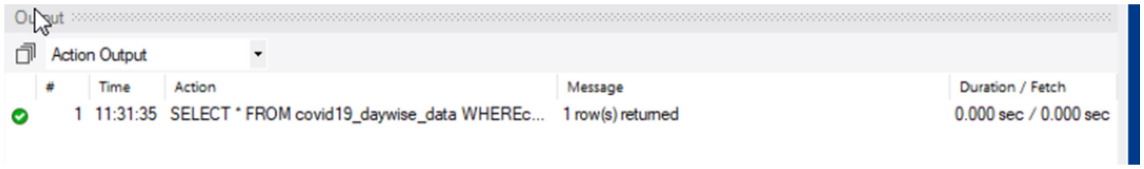




Let's run the same query again:

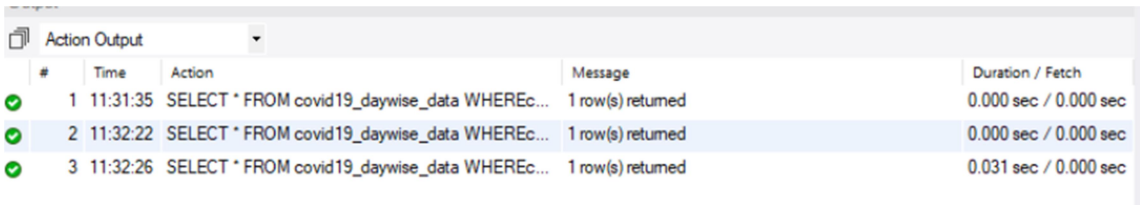


We can even run it again to make sure the improvement on the duration of the execution was not actually a glitch:



Now is so fast is not even showing a duration!

If we run it few more times we will see that sometimes this duration can slightly increase, but in no instance gets the original value, still much much quicker :



This may not sound like a lot, but when you consider millions of queries, it is a major improvement!

Task 4: Create a summary database table for specific data

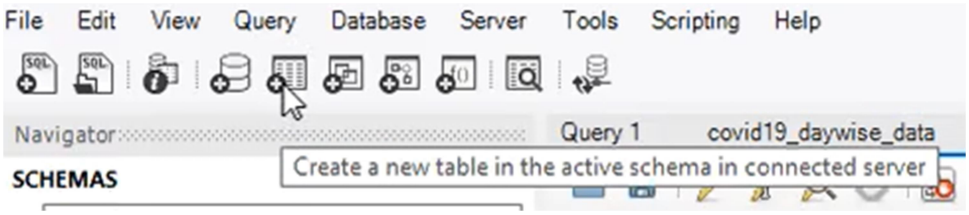
In this task we will create a summary database table to hold specific data that users may need over and over again.

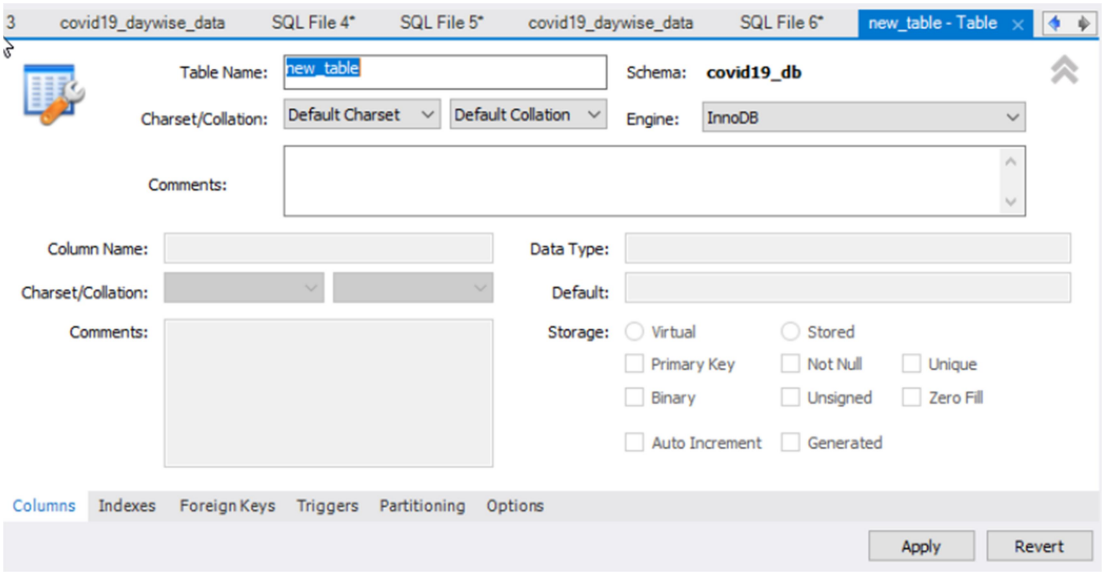
By doing this they can quickly obtain the data they need without having to do any complex queries which also translates in a much better performance as well as will contain only the required information instead of fetching the whole database.

This technique is often used in data warehousing

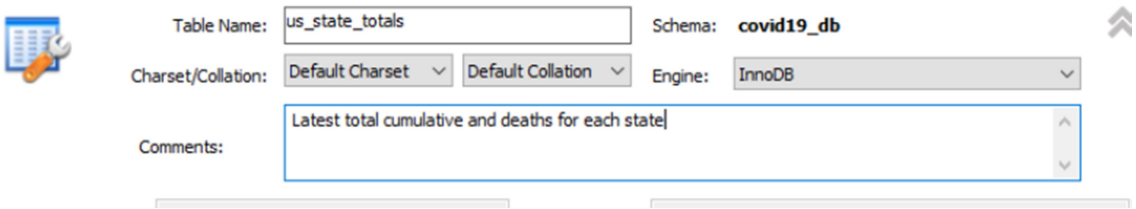
As a note, to mention that is important to pay attention to the names we use so the user knows where the data came from the original table.

So let's create a new table within the database. This table will house the latest totals for each US state

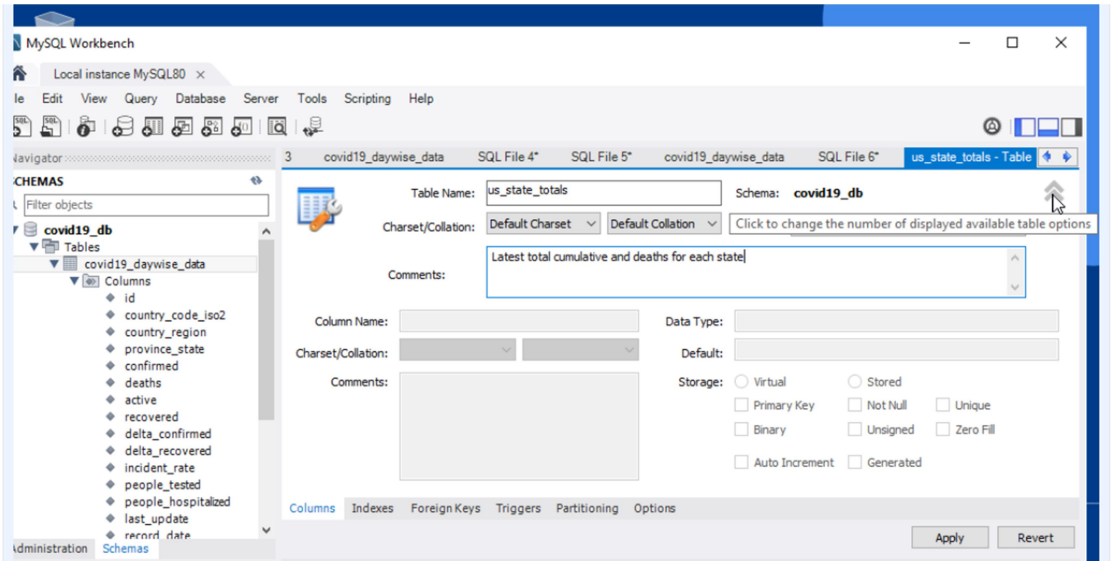




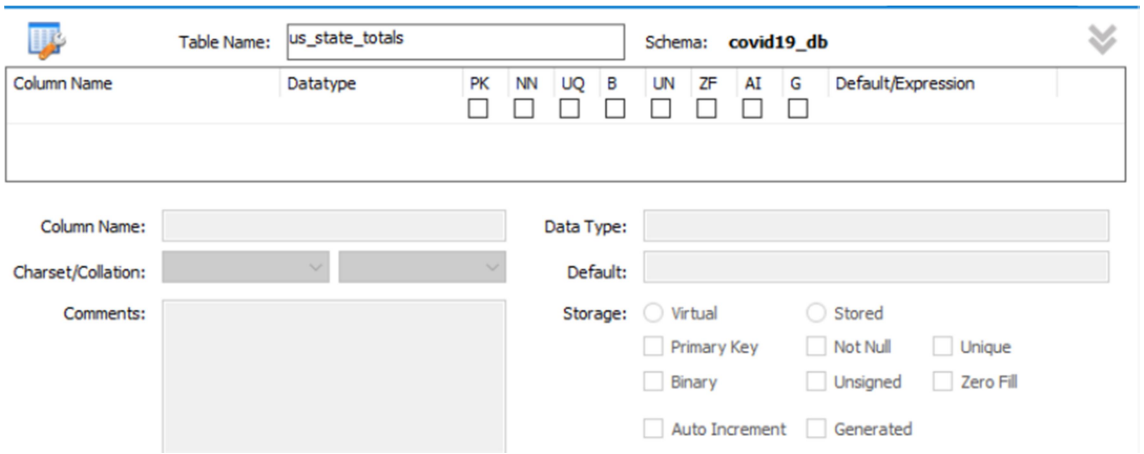
Let's call it us_state_totals and we can add some comments so Users know what this table is:



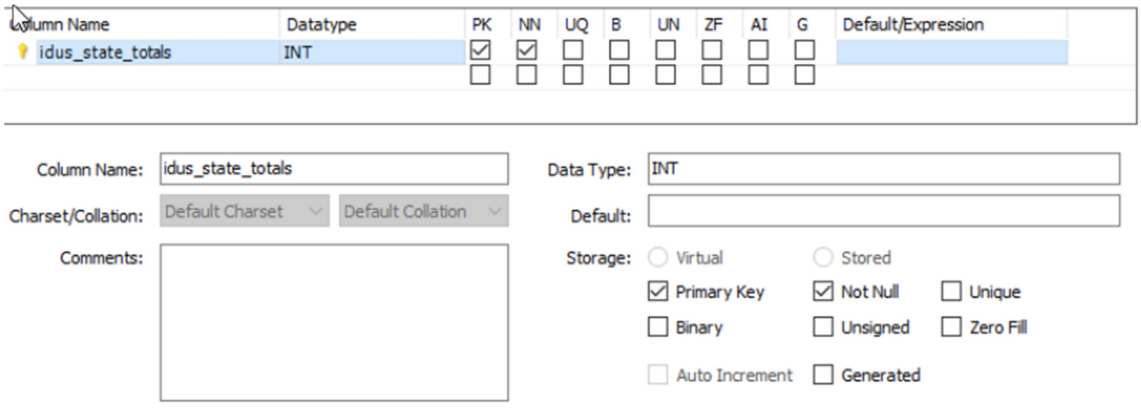
If by default we don't have the column options we can click in the double arrow to bring them:



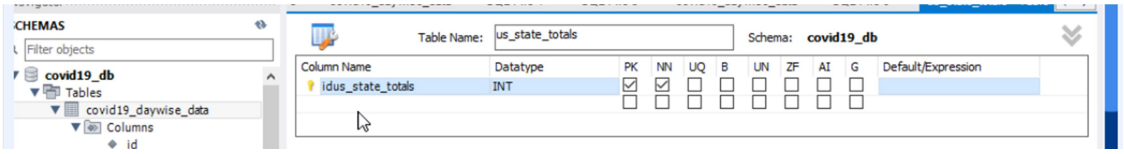
Here is what we will see:



We check the box for PK (Primary Key) which automatically clicks NN (Not Null) too:

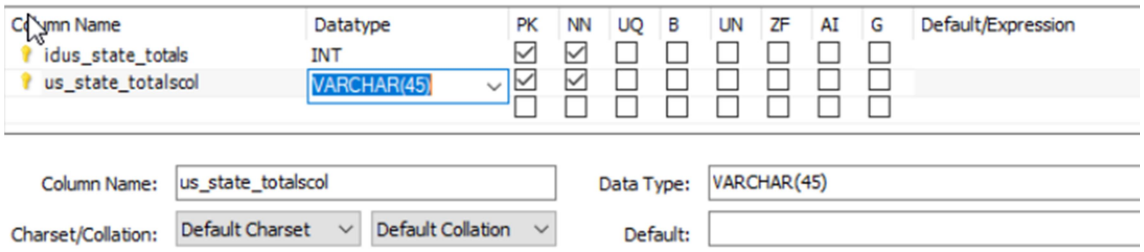


Now, we add another one by clicking under the column name:



And this appears:

We can now click in VARCHAR and bring it into the table:



We can change the length to 255:

And by clicking in the Column name on the above panel, we can change the name to give it the same name than the main data base:

Now under Province State, let's click again and create a confirmed field which will make BIGINT instead of VARCHAR.

Deaths, which will also be a BIGINT

And Record_date as DATE

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
confirmed	BIGINT	☐	☐	☐	☐	☐	☐	☐	☐	
deaths	BIGINT	☐	☐	☐	☐	☐	☐	☐	☐	
record_date	DATE	☐	☐	☐	☐	☐	☐	☐	☐	

Note: we need to realize that one of the pitfalls of handling large data sets with numeric values such as totals is making certain proper data types are used for totals.

If data type INT is used for example, it may go beyond its range.

So it's a good idea to use BigINT for totalling data that is constantly growing in large tables such as this.

Well, now the fields already match the names in the DB, let's apply the changes (which will show us the summary of what we have just done):

Review the SQL Script to be Applied on the Database

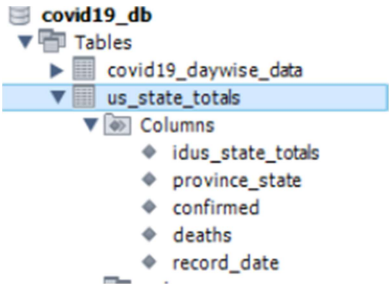
Online DDL
Algorithm: Default Lock Type: Default

```
1  CREATE TABLE `covid19_db`.`us_state_totals` (
2    `idus_state_totals` INT NOT NULL,
3    `province_state` VARCHAR(255) NULL,
4    `confirmed` BIGINT NULL,
5    `deaths` BIGINT NULL,
6    `record_date` DATE NULL,
7    PRIMARY KEY (`idus_state_totals`))
8  COMMENT = 'Latest total cumulative and deaths for each state';
9
```

If all looks good, let's apply and finish.

Now let's click in the Refresh button on SCHEMAS:

And now we can see our new table with its columns:



Task 5: Populate the summary table using SQL

We will go through the following steps:

a. Use INSERT statement for the summary table
b. Use the SELECT statement to gather the corresponding data from the original table
c. Add DELETE before the INSERT to refresh rows when main table updates
d. Sufficient for infrequent updates

Note that Point b, will be easy to map the two fields together because we already made the field names match.

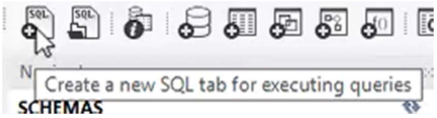
Note for Point c, we'll delete the data from the table upon each update because the updates only occur on a daily basis

If it was getting updated constantly then we would use an update statement but we can just delete the data and refresh the rows when they're when the updates occur

Let's get started:

So in order to populate the summary table we need to use an insert command that has the summary table as a target with the values that we need to insert into the columns.

And then we'll use the original table as the source of the data for the columns

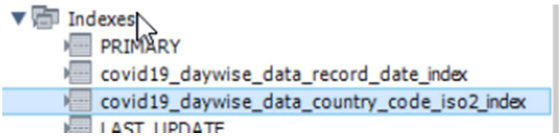


Let's use the INSERT statement on all our columns:

```
INSERT INTO us_state_totals (province_state, confirmed, deaths, record_date)
```

Now we need to do is get our select statement together, insert values from the original database.

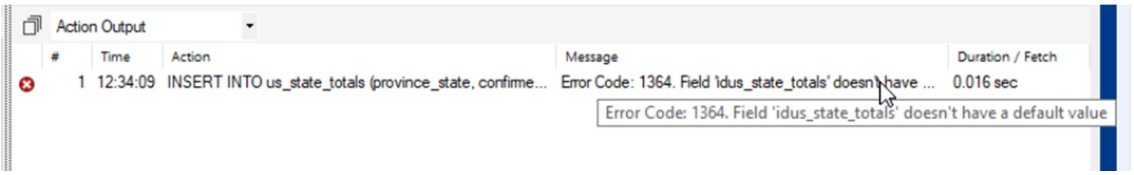
For country_region, we can use country_code_iso2 instead because is one of the indexes (now that we know what an INDEX is makes much more sense, doesn't it?) , so even this runs only once a day, will be more efficient (we have to aim to gain efficiency whenever we can! At the end, all sums up)



So our statement would be:

```
INSERT INTO us_state_totals (province_state, confirmed, deaths, record_date)
SELECT province_state, confirmed, deaths, record_date FROM covid19_daywise_data
WHERE province_state is not NULL
    AND province_state <> ''
    AND country_code_iso2 = 'US'
    AND record_date = (SELECT MAX(record_date) FROM covid19_daywise_data);
```

We run the query but we encounter the following error

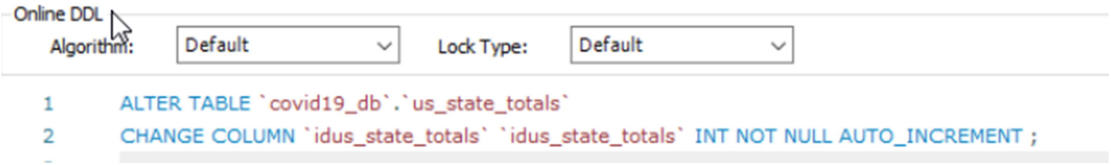


Let's fix that!

AI (Auto Increment) was not checked, so check it and apply

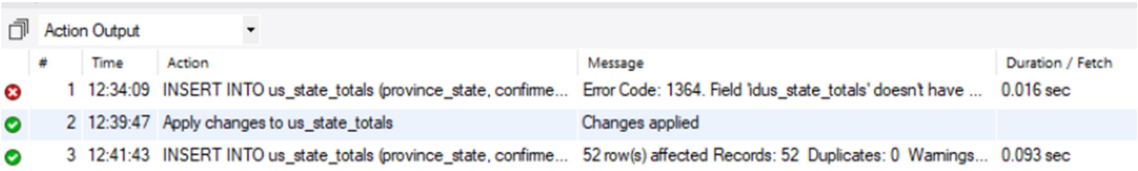
Remember, a summary will pop out:

Review the SQL Script to be Applied on the Database



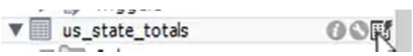
Apply again and Finish

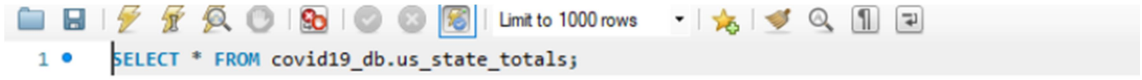
Now we can go back to our INSERT and re-run it:

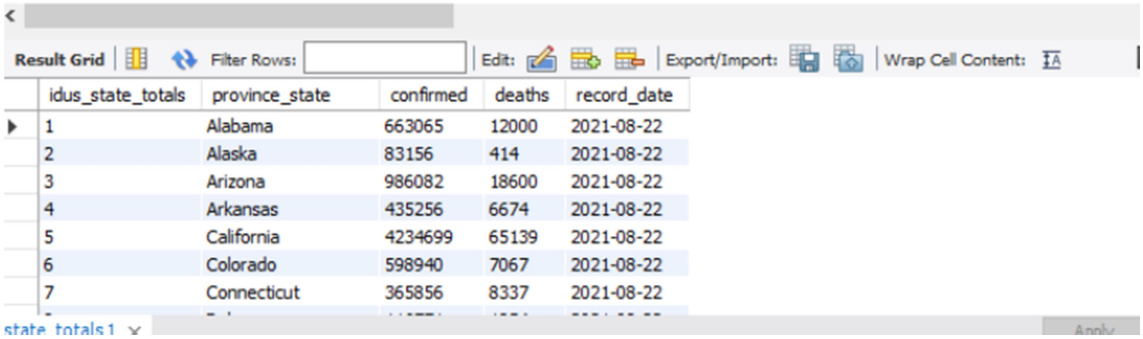


We have 52 rows affected. Which is what is expected, so it's looking good.

Now let's query our state totals





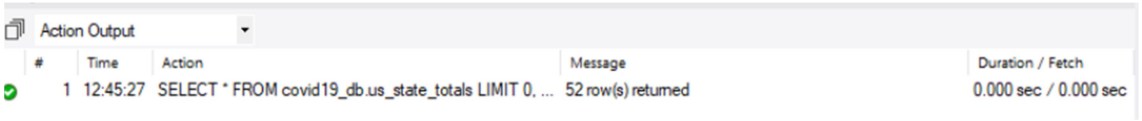


Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

idus_state_totals	province_state	confirmed	deaths	record_date
1	Alabama	663065	12000	2021-08-22
2	Alaska	83156	414	2021-08-22
3	Arizona	986082	18600	2021-08-22
4	Arkansas	435256	6674	2021-08-22
5	California	4234699	65139	2021-08-22
6	Colorado	598940	7067	2021-08-22
7	Connecticut	365856	8337	2021-08-22

state_totals1

And now let's run it to see the performance:



Action Output

#	Time	Action	Message	Duration / Fetch
1	12:45:27	SELECT * FROM covid19_db.us_state_totals LIMIT 0, ...	52 row(s) returned	0.000 sec / 0.000 sec

Is very fast and now, anyone wanting to get the latest state data can run this very simple query by clicking a couple of times and get all the required data in no time.

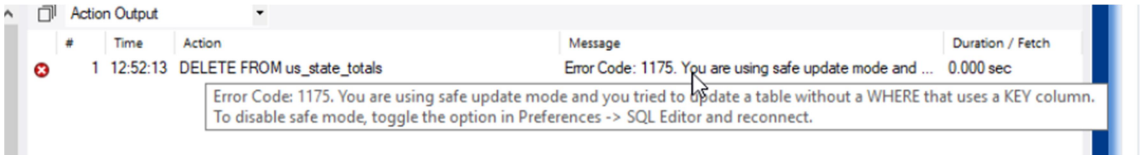
So now if we wanted to update the summary table every time the original table is updated we could DROP and recreate the table each time but that takes a lot of effort.

We could also come up with an UPDATE SQL but since it's only update on a daily basis we can simply DELETE the data from the table and start over.

So in our query we're going to add a DELETE before our INSERT statement:

```
1  DELETE FROM us_state_totals;
2  INSERT INTO us_state_totals (province_state, confirmed, deaths, record_date)
3  SELECT province_state, confirmed, deaths, record_date FROM covid19_daywise_data
4  WHERE province_state is not NULL
5      AND province_state <> ''
6      AND country_code_iso2 = 'US'
7      AND record_date = (SELECT MAX(record_date) FROM covid19_daywise_data);
```

But we get an error:



Action Output

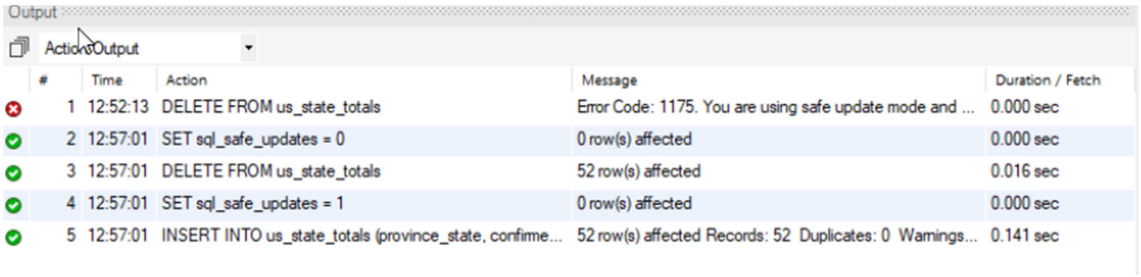
#	Time	Action	Message	Duration / Fetch
1	12:52:13	DELETE FROM us_state_totals	Error Code: 1175. You are using safe update mode and ...	0.000 sec

Error Code: 1175. You are using safe update mode and you tried to update a table without a WHERE that uses a KEY column.
To disable safe mode, toggle the option in Preferences -> SQL Editor and reconnect.

To get around this issue, since we know we want to do this, each time we're going to add a couple of turn safe update mode off before the delete statement and then turn it back on again after the delete statement:

```sql
SET sql_safe_updates = 0;
DELETE FROM us_state_totals;
SET sql_safe_updates = 1;
INSERT INTO us_state_totals (province_state, confirmed, deaths, record_date)
SELECT province_state, confirmed, deaths, record_date FROM covid19_daywise_data
WHERE province_state is not NULL
    AND province_state <> ''
    AND country_code_iso2 = 'US'
    AND record_date = (SELECT MAX(record_date) FROM covid19_daywise_data);
```

And we run it again:



Now we can run it every time our updates occur

So what have we learnt in here:

1. How get around MySQL and execute queries

    - Check data types
    - Check the engine that is being used
    - Verify the data values are correct by doing queries measuring the data from different point of views to check if results match.
    -

2. Create INDEXES to improve query performance

3. Create Summary tables for specific & repetitive queries

    - Improve performance
    - Good practices on column names for easy identification of the origin of the data.
    - Simplifies the process for the user.