Introduction, Queries and Summary of the Report

Welcome to an analysis of Covid-19 and it's spread from early 2020 to early 2021. I have made this Data Warehouse to provide dynamic answers to pressing business queries. I have followed the following design steps, which are explained within this document:

1. What are the concept hierarchies for our dimension tables?
2. Deciding Starnet for our Data Warehouse design.
3. ETL
4. Using SQL to implement our Data Warehouse design
5. What does the database diagram for our Data Warehouse look like? Using SQL Server Management Studio to make a diagram.
6. Using SQL Server Data Tools to build a multi-dimensional analysis solution, with a cube designed to answer queries.
7. Using Power BI to Visualize the data from your business queries.
8. Going through all the OLAP Operations used to answer all the queries.

Business Queries answered with this report.

1. What is the total number of confirmed cases in Australia in 2020? What is the number of confirmed cases in each quarter of 2020 in Australia? What is the number of confirmed cases in each month of 2020 in Australia?
2. In Sept 2020, how many recovered cases are there in the region of the Americas? How many recovered cases in the United States, Canada and Mexico, respectively, in Sep 2020?
3. What is the total number of covid deaths worldwide in 2020? What is the total number of covid deaths in large countries, medium countries and small countries, respectively, in 2020?
    - **Note**: In this project, country size is measured by population. Large countries: population >= 40 million; small countries: population <= 2 million; medium countries: 2 million < population < 40 million.
4. Do countries with a life expectancy greater than 75 have a higher recovery rate?

Design, Implementation and Concept Hierarchies

In order create our StarNet model we first had to build a concept hierarchy for each dimension. Each concept hierarchy is of total order with the value "ALL" in the root node as shown in the figure.

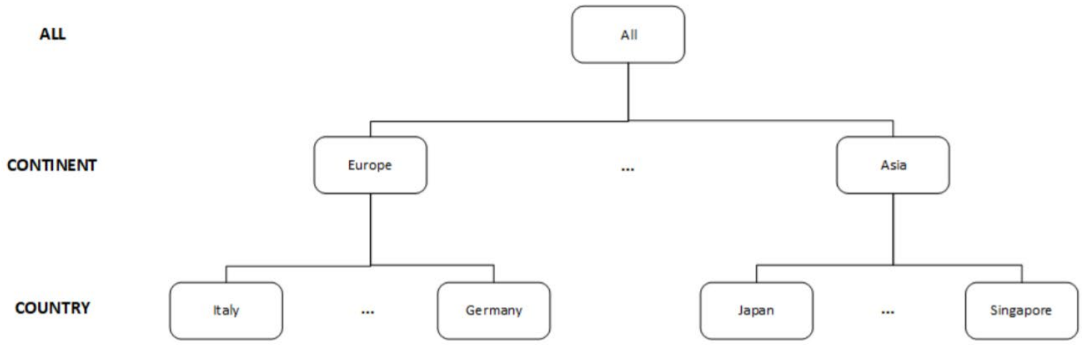


Figure1.1: Concept Hierarchy for Location

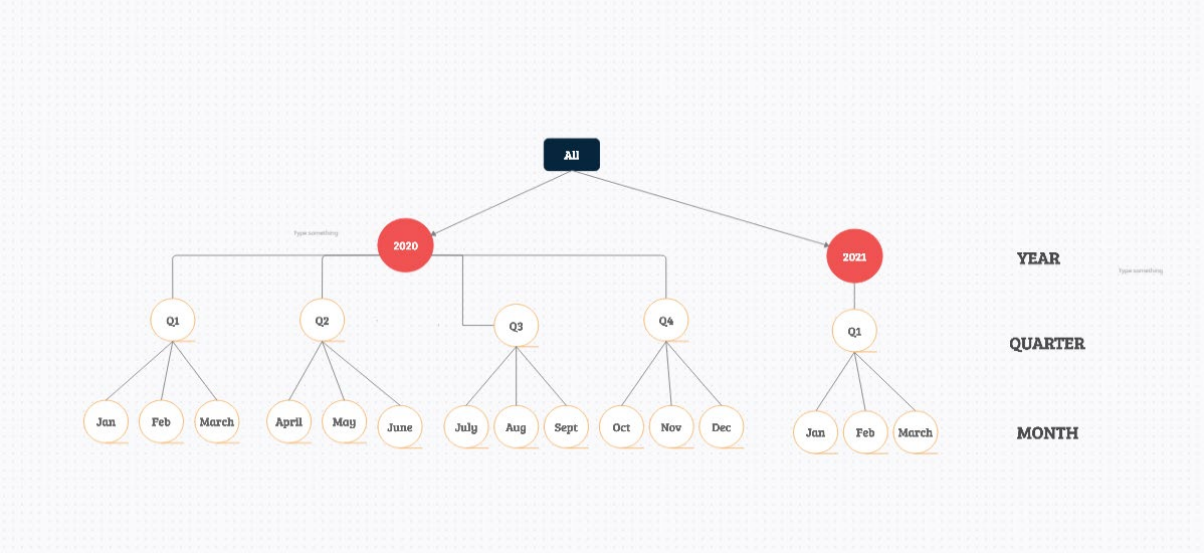


Figure1.2: Concept Hierarchy for Time

Each level in the concept hierarchy is then used in building the StarNet model.

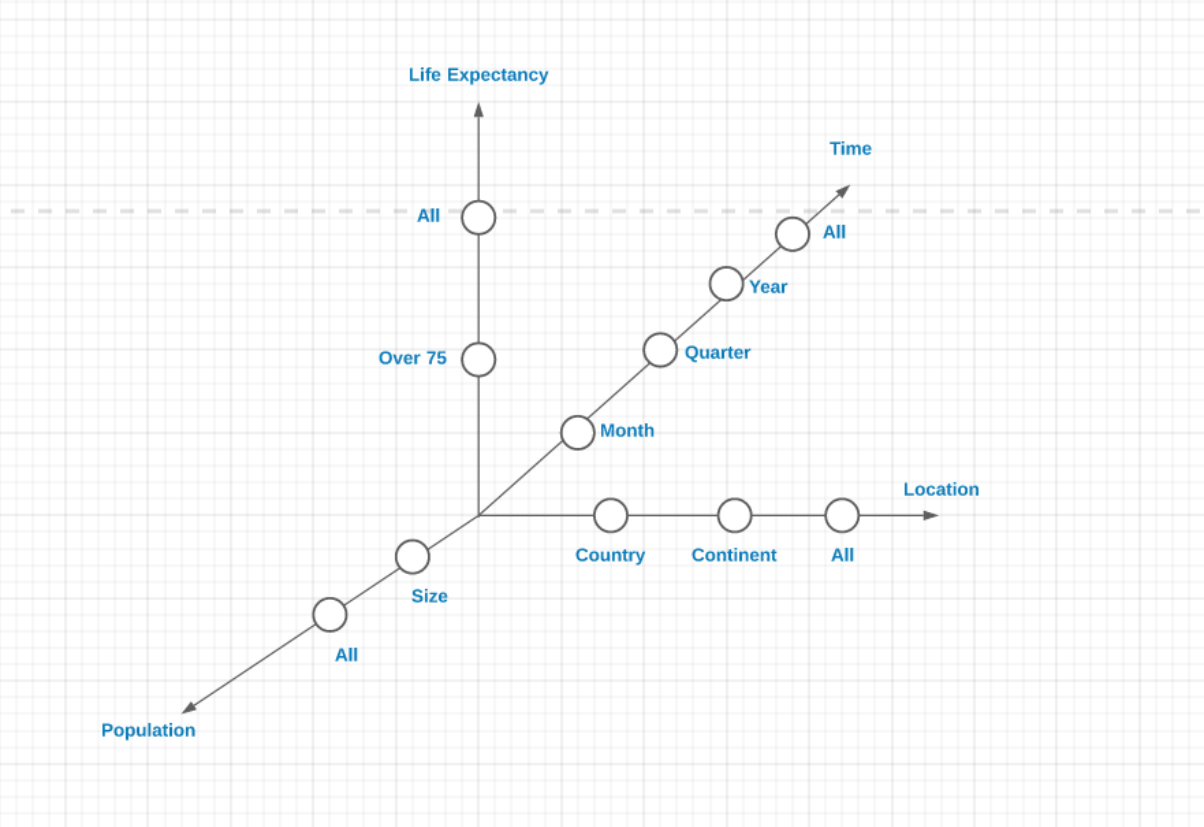


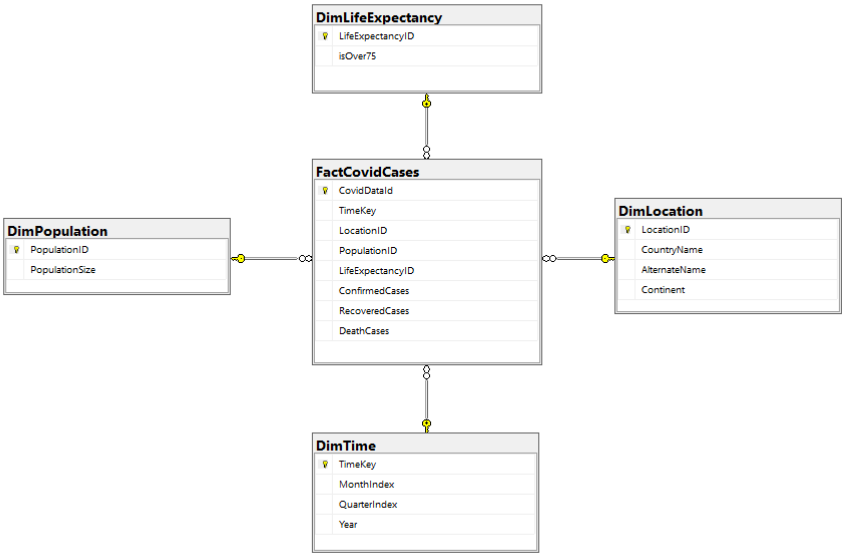
Figure1.3: STARNET MODEL

We had to use both low level and high-level information to answer all our queries. There were a few different models that I came up with along the way, but I decided to stick with this because it answers all our queries comfortably. Some StarNet Footprints for the queries look like:

DataBase Schema

Our Database follows the structure of Star Schema. Rather than representing population and life expectancy as a part of location, we used them separately. Our Fact table will have direct

relationship with population(dimPopulation) and life expectancy (dimLifeExpectancy) with this approach. The idea behind this was so that Population and Life Expectancy can be grouped and can be visually represented in a group. This has also proven to be useful when generating the data cube. Each table has its primary key which acts as a foreign key in fact table.



ETL

The ETL process was done using python code. I have used multiple pythons. Each one is responsible for creating a table (whether dimension/fact table). The reason for doing this was to avoid changing content of a file while not intending to. The other way of doing this was to use Object-Oriented Programming approach and making several few smaller functions.
I started with extracting all the locations from the data.

```
data = pd.read_csv("time_series_covid19_confirmed_global.csv") #reads csv file
data = data.fillna(0) #Fills in 0 for every empty location in the dataframe
data_countries = data['Country/Region'] #collecting the list of countries from times_series files.
data_countries = data_countries.drop_duplicates() #dropping any repitition
data = data.groupby(['Country/Region']).sum()
data = data[data.columns[2:]]
data2 = pd.read_csv("owid-covid-data.csv") #reads owid data file and stores it in data2
data2 = data2.fillna(0)
data2 = data2[['location', 'continent']]
data2 = data2.drop_duplicates()
```

From this I collected country/location names in two separate Dataframes.

```
data3 = pd.merge(data, data2, how='inner', left_on='Country/Region', right_on='location')
data4 = pd.merge(data, data2, how="outer", left_on="Country/Region", right_on="location")


data2_loc = data2['location']
data2_removed_countries = data2[~data2['location'].isin(data3['location'])]
```

I merged both datas using inner and outer joins. Inner join helps in removing any duplicates and keeps the data with common countries in both files. While outer join helps in keeping a dataframe for all the countries, even if one of them is missing from another. I stored all the locations that were present in "owid_covid-data" file inside data2_removed_countries dataframe. Similarly, I got a Dataframe all the countries present time series files which aren't present in both the files.
These were the country names which were present in owid-covid-data file but not in time-series files:

	location	continent
10076	Cape Verde	Africa
12352	Congo	Africa
14627	Democratic Republic of Congo	Africa
24052	Hong Kong	Asia
25423	International	0
38121	Myanmar	Asia
42021	Palestine	Asia
51446	South Korea	Asia
54373	Taiwan	Asia
55699	Timor	Asia
58624	United States	North America
59925	Vatican	Europe
60900	World	0

While these were present in time series and not in Owid-covid-data file.

34	Burma
36	Cabo Verde
93	Congo (Brazzaville)
94	Congo (Kinshasa)
104	Diamond Princess
142	Holy See
158	Korea, South
171	MS Zaandam
182	Micronesia

Now, through inspection we find out that some of these names are for the same country. It's just that they are spelled differently. So, the best way to account for this was to add these files back to the list of countries/locations. The following code adds them back into the pandas dataframe of countries.

```
data3 = data3.append(data4[data4['location'] == 'United States'])
data3 = data3.append(data4[data4['location'] == 'Cape Verde'])
data3 = data3.append(data4[data4['location'] == 'Congo'])
data3 = data3.append(data4[data4['location'] == 'Democratic Republic of Congo'])
data3 = data3.append(data4[data4['location'] == 'Myanmar'])
data3 = data3.append(data4[data4['location'] == 'South Korea'])
data3 = data3.append(data4[data4['location'] == 'Vatican'])
data3 = data3.append(data4[data4['location'] == 'Taiwan'])
data3 = data3.append(data4[data4['location'] == 'Palestine'])
data3 = data3.append(data4[data4['location'] == 'Timor'])
```

Now, because some of the countries have multiple names, I though it would be a good idea to keep a column for alternate names in our dimension table for countries.

```
data3["alternate_location"] = ""
data3["alternate_location"] = np.where(data3['location'] == "United States", "US", data3["alternate_location"])
data3["alternate_location"] = np.where(data3['location'] == "Cape Verde", "Cabo Verde", data3["alternate_location"])
data3["alternate_location"] = np.where(data3['location'] == "Myanmar", "Burma", data3["alternate_location"])
data3["alternate_location"] = np.where(data3['location'] == "Congo", "Congo (Brazzaville)", data3["alternate_location"])
data3["alternate_location"] = np.where(data3['location'] == "Democratic Republic of Congo", "Congo (Kinshasa)", data3["alternate_location"])
data3["alternate_location"] = np.where(data3['location'] == "Vatican", "Holy See", data3["alternate_location"])
data3["alternate_location"] = np.where(data3['location'] == "South Korea", "Korea, South", data3["alternate_location"])
data3["alternate_location"] = np.where(data3['location'] == "Taiwan", "Taiwan*", data3["alternate_location"])
data3["alternate_location"] = np.where(data3['location'] == "Timor", "Timor-Leste", data3["alternate_location"])
data3["alternate_location"] = np.where(data3['location'] == "Palestine", "West Bank and Gaza", data3["alternate_location"])
```

Now, a few of the names were not added to the data even after this because a few of them were ships and not a country. That's why I decided to skip on names like 'Diamond Princess' and 'MS Zaandam'. I decided to not include Hong Kong to our data because of the inconsistency of the data present in Hong Kong.

This gave us our location table:

```
locations = data3[['location', 'alternate_location', 'continent']].reset_index().reset_index()
locations = locations[['location', 'alternate_location', 'continent']]
locations.to_csv("locations.csv", header=False)
print(locations)
```

Note: Print statement is only for self-checking. .to_csv is used to write a csv file.

To extract a csv file for time, I first of all made sure that all of the three time_series csv files given to us had the same dates being used.

```
import pandas as pd

data = pd.read_csv("time_series_covid19_deaths_global.csv")
headers = list(data.columns.values)
headers = headers[4:]

data2 = pd.read_csv("time_series_covid19_recovered_global.csv")
headers2 = list(data2.columns.values)
headers2 = headers2[4:]


data3 = pd.read_csv("time_series_covid19_confirmed_global.csv")

headers3 = list(data3.columns.values)
headers3 = headers3[4:]
```

These lists matched each other which meant that all time_series csv files have same dates.
Next, I extracted a Dataframe of all the dates in owid-covid file. The following code also converts the Object type of data to datatime type which helped me in comparing and using dates.

```
data_owid = pd.read_csv("owid-covid-data.csv")
data_owid = data_owid['date']
data_owid = pd.DataFrame(data_owid)

data_time_series = pd.DataFrame(headers, columns=["dates"])
data_owid = data_owid.drop_duplicates()

data_time_series["dates"] = pd.to_datetime(data_time_series["dates"])

#print(data_time_series.dtypes)
data_owid.columns = ["dates"]
data_owid["dates"] = pd.to_datetime(data_owid["dates"])
#print(data_owid.dtypes)
dates_extra = data_time_series[~data_time_series.isin(data_owid)]
```

Next, I concatenated dataframes that I got from both files which game me time_series that I stored in a csv file here.

```
frames = [data_owid, data_time_series]
result = pd.concat(frames)
result = result.drop_duplicates()
result.to_csv("times.csv", header=False)
result['month'] = pd.DatetimeIndex(result['dates']).month
result['quarter'] = pd.DatetimeIndex(result['dates']).quarter
result['year'] = pd.DatetimeIndex(result['dates']).year

result = result[['month', 'quarter', 'year']]
result = result.drop_duplicates()
result = result.reset_index()
result = result[['month', 'quarter', 'year']]
#print(result)
result.to_csv("times.csv", header=False)
```

Next, I wrote a csv file for life expectancy using Python. There were only two possibilities here. This could have been done manually but I decided to write it through code to be consistent.

```
import pandas as pd
data_owid = pd.read_csv("owid-covid-data.csv") #Reads the file
data_loc = pd.read_csv("locations.csv", names=['index', 'Country', 'Alternate Name', 'Continent']) #Reads file locations which we have just
data_loc = data_loc[['index', 'Country', 'Continent']] #GEtting only the tables required
data_owid = data_owid[['location', 'life_expectancy']].drop_duplicates() #Getting only the required data and dropping duplicates
data_life = pd.merge(data_owid, data_loc, how='inner', left_on='location', right_on='Country') #Can use inner join here because we know tha
data_life = data_life.fillna(0)
data_life['greater_than_75'] = data_life['life_expectancy'] > 75 #Checking whether the life expectancy in countries is more than 75 or not
#data_life = data_life[['index', 'greater_than_75']] #Keeping only required columns for our queries
#data_life.to_csv("life_expectancy.csv", header=False) #

data_life_updated = data_life[['greater_than_75']] #Keeping only required columns for our queries
data_life_updated = data_life_updated.drop_duplicates()
data_life_updated.to_csv("actual_life_expectancy.csv", header=False)
```

To implement fact Table and to extract data for it, I first of all gathered all the data that I will need. It was important to have necessary data for the integration of our csv files.

```
import pandas as pd
import numpy as np
data_owid = pd.read_csv("owid-covid-data.csv")
data_deaths = pd.read_csv("time_series_covid19_deaths_global.csv")
data_confirmed = pd.read_csv("time_series_covid19_confirmed_global.csv")
data_recovered = pd.read_csv("time_series_covid19_recovered_global.csv")
times = pd.read_csv("time.py", names=["time_index", "month", "quarter", "year"])
locations = pd.read_csv("locations.csv", names=['loc_index', 'Country', 'Alternate Name', 'Continent'])
life_exp = pd.read_csv("life_expectancy.csv", names=["life_index", "life_country_index", "greater_than_75"])

population = pd.read_csv("population.csv", names=["pop_index", "pop_loc_index", "pop_size"])
real_pop = pd.read_csv("actual_population.csv", names=["population_index", "population_size"])
```

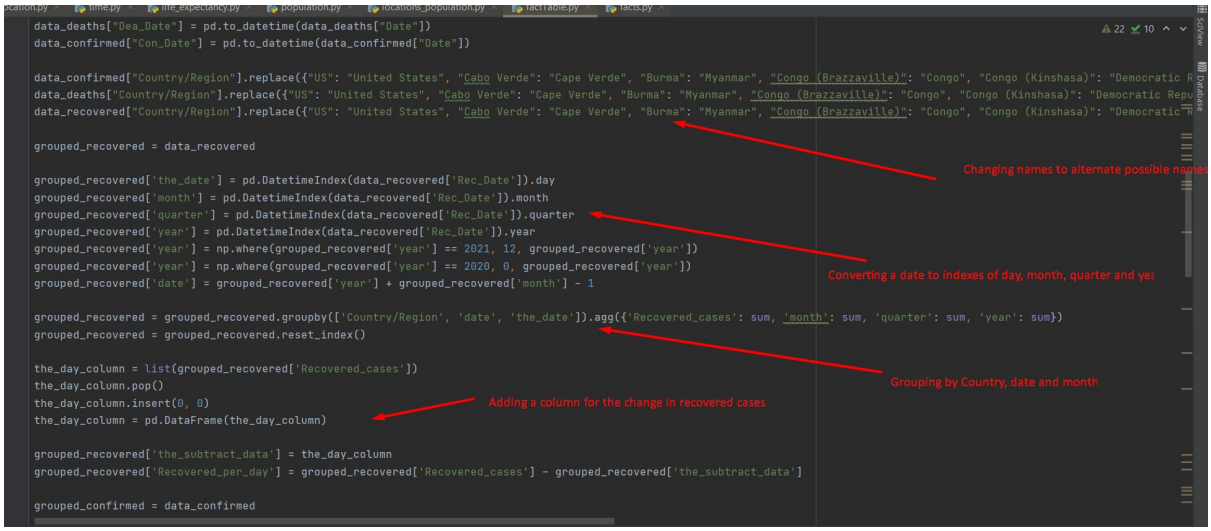
Then I merged population file and life expectancy files to get their indexes in our fact table file

```
data_owid_loc = pd.merge(data_owid, locations, how='inner', left_on='location', right_on='Country')
data_owid_pop_loc = pd.merge(data_owid_loc, population, how='inner', left_on='loc_index', right_on='pop_loc_index')
data_owid_pop_loc = pd.merge(data_owid_pop_loc, real_pop, how='inner', left_on='pop_size', right_on='population_size')

real_life_exp = pd.read_csv("actual_life_expectancy.csv", names=['life_exp_index', 'over_75'])
data_owid_pop_loc_life = pd.merge(data_owid_pop_loc, life_exp, how='inner', left_on='loc_index', right_on='life_country_index')
data_owid_pop_loc_life = pd.merge(data_owid_pop_loc_life, real_life_exp, how='inner', left_on='greater_than_75', right_on='over_75')
```

```
data_recovered = data_recovered.melt(id_vars=["Province/State", "Country/Region", "Lat", "Long"],
        var_name="Date",
        value_name="Recovered_cases")
data_deaths = data_deaths.melt(id_vars=["Province/State", "Country/Region", "Lat", "Long"],
        var_name="Date",
        value_name="Death_cases")
data_confirmed = data_confirmed.melt(id_vars=["Province/State", "Country/Region", "Lat", "Long"],
        var_name="Date",
        value_name="Confirmed_cases")
```

It was important to have access to the lowest level of information and that is why the cumulative data given in our files wasn't sufficient. We needed data for every day for drill down and roll up functioning. In this code I have entered a table to enter recovered cases to our file.



There were multiple values in our code which turned out to be negative. This suggests a reduction in the number of total cases which is simply not possibly. This could be an error of two varieties. One of them could be an error in recording the data. For example, some of the sources might be publishing a different data than others. This should not be too huge and can be neglected as it will balance itself out over the time. The other kind of error could be human error, i.e. some error by humans in entering the data. This is the error that we must remove from our data as this could be huge sometimes. The following code removed it for me.

Implementation

The Covid_data_DW was implemented using SQL Script.

The script is responsible for creating the appropriate dimensions and fact tables, setting up constraints, and referencing all primary and foreign keys accordingly. Prior to this, any existing Covid_data_DW databases are automatically dropped and re-created.

```
GO
IF EXISTS (SELECT [name] FROM [master].[sys].[databases] WHERE [name] = 'COVID-DATA')
DROP DATABASE CovidDW;
GO
PRINT '';
PRINT '*** Creating Database';
GO
Create database CovidDW
Go
Use CovidDW
Go

PRINT '';
PRINT '*** Creating Table DimLocation';
GO
Create table DimLocation
(
LocationID int primary key,
CountryName varchar(50) not null,
AlternateName varchar(50),
Continent varchar(50) not null,
)
Go
```

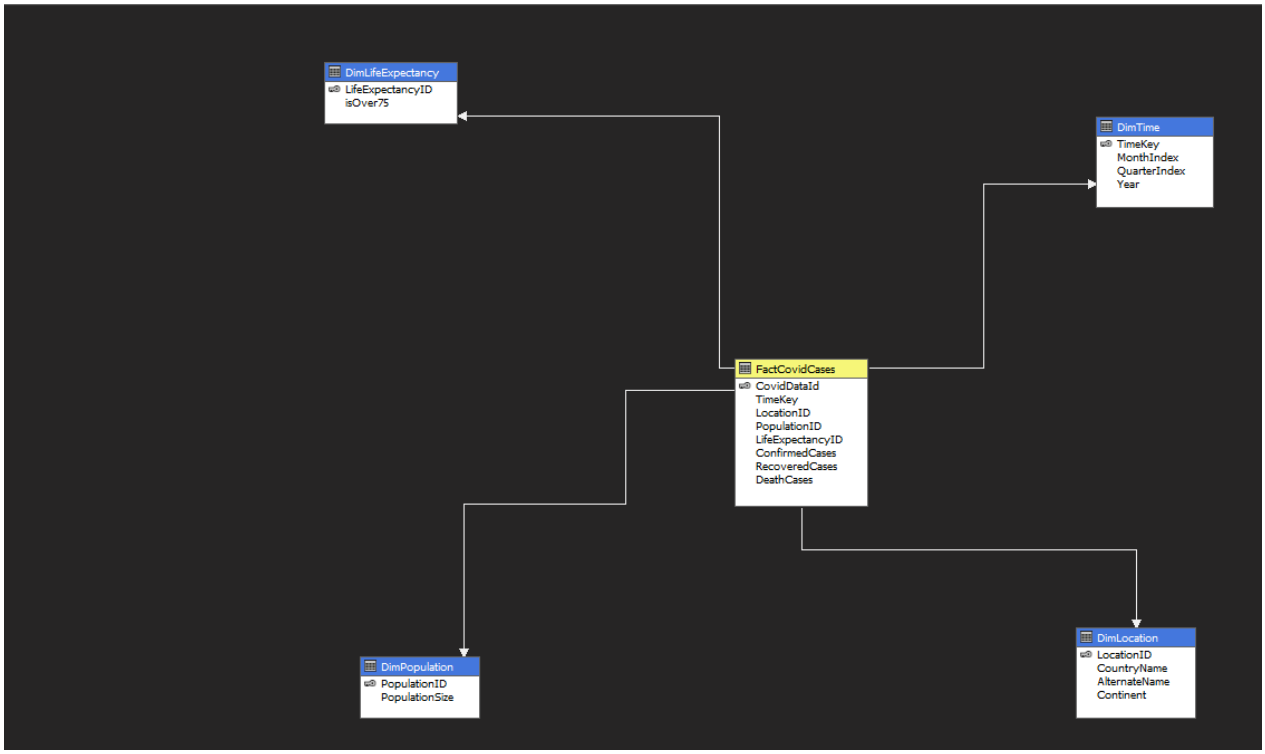
Foreign keys constraints are declared accordingly to ensure that data inserted into the tables are valid.

```
BULK INSERT [dbo].[DimLifeExpectancy] FROM 'C:\Users\tunaa\Downloads\covid-data(1)\covid-data\actual_life_expectancy.csv'
WITH (
    CHECK_CONSTRAINTS,
    --CODEPAGE='ACP',
    DATAFILETYPE='char',
    FIELDTERMINATOR=',',
    ROWTERMINATOR='\n',
    --KEEPIDENTITY,
    TABLOCK
);
```

The Covid_Data_DW database can be populated with the appropriate data using the populate SQL script. An example of which is given above.

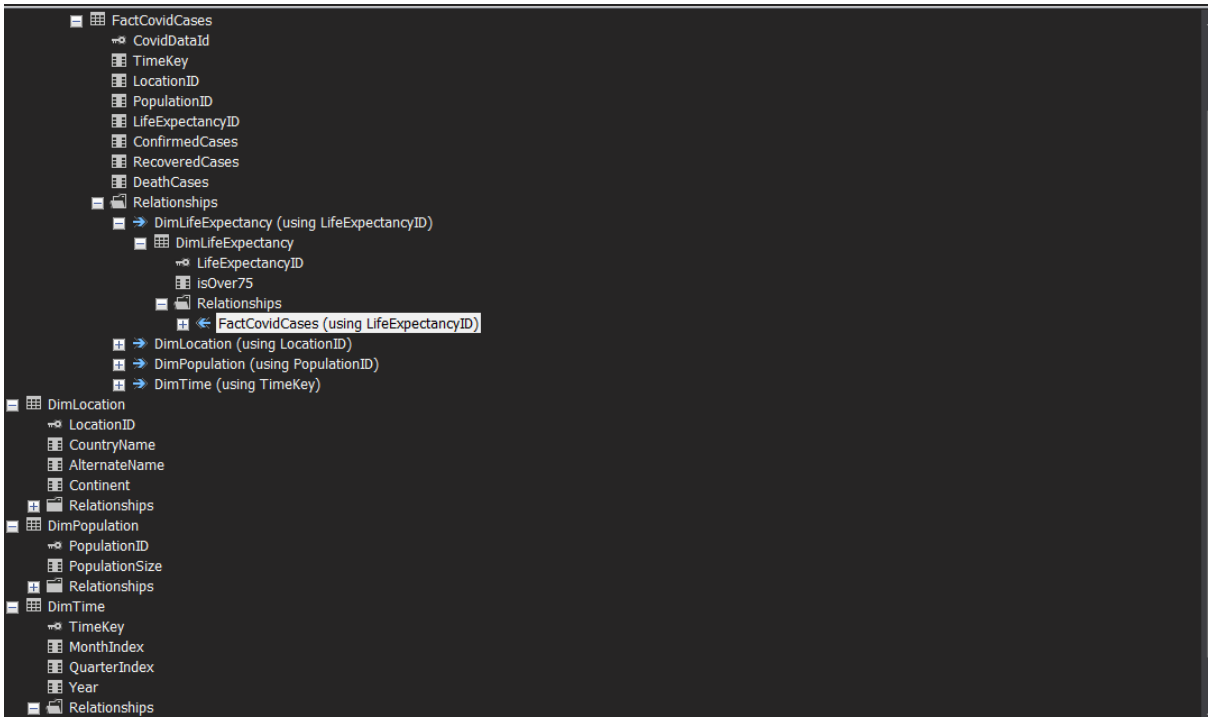
We must enter the data in dimension tables containing primary keys before entering data in the fact table with the foreign key.

Visualisation



The next in this process is Visualisation. Visualising Covid_data_DW as a data cube allows for Roll Up and Drill Down operations, making our business queries a lot more versatile, and explores information that normally would not be found without a data cube. Furthermore, creating hierarchies allows up to drill down further into the data, finding information that can prove useful in a business scenario.

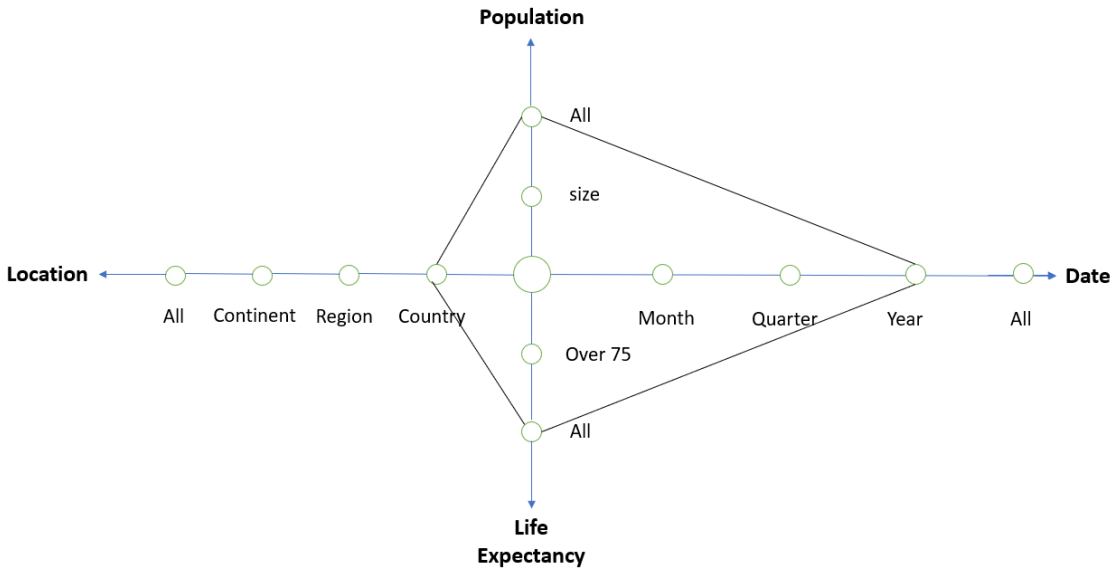
Additionally, the cube needs to have the data loaded into the server using SSMS, before it can be deployed for use by PowerBI.



Query 1

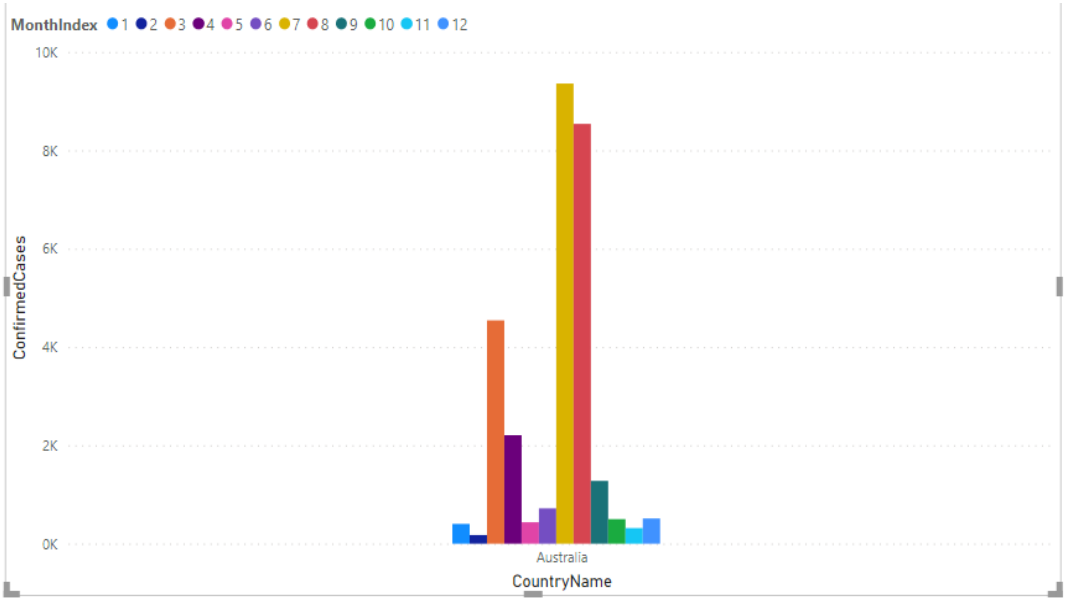
What is the total number of confirmed cases in Australia in 2020? What is the number of confirmed cases in each quarter of 2020 in Australia? What is the number of confirmed cases in each month of 2020 in Australia?

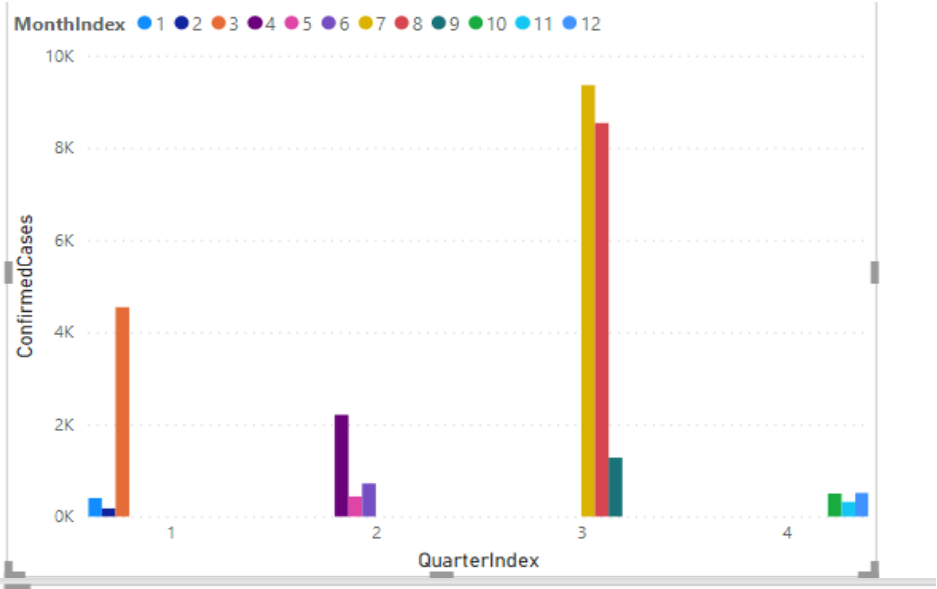
To answer this question, we can first consider our StarNet Model. This gives us a good clarity of what's needed. Although, this will change depending on the specific part of the query.



Our first business query demonstrates the use of the Drill Down and Roll Up operations on our PowerBI visualisations.

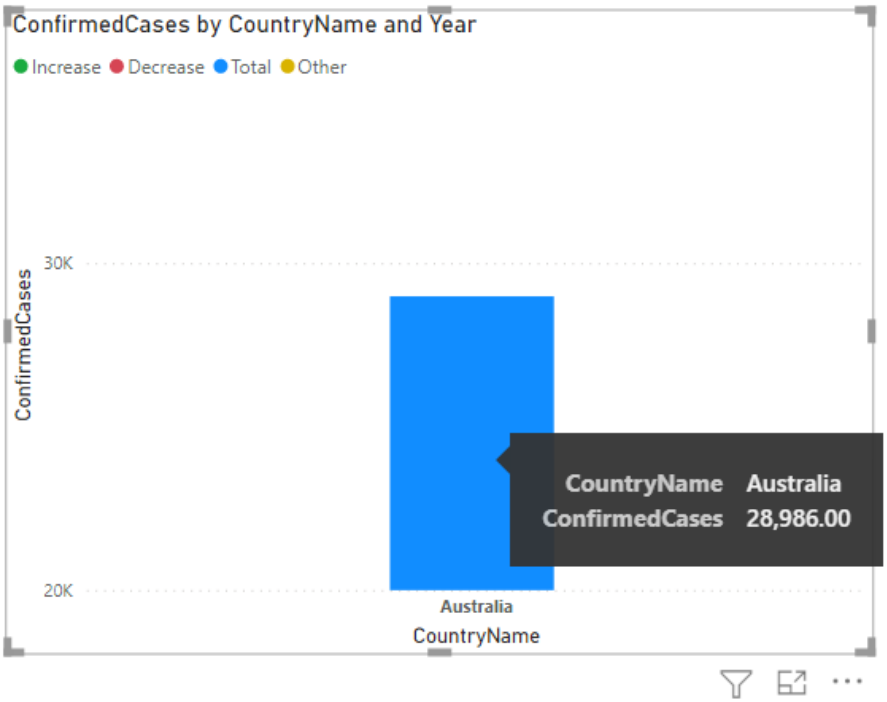
Illustrated below are to bar graphs for number of confirmed cases in Australia in 2020. This graph has the ability to Drill Down to a lower level, which illustrates the number of confirmed cases for each selected level of time (month and quarter in these two)





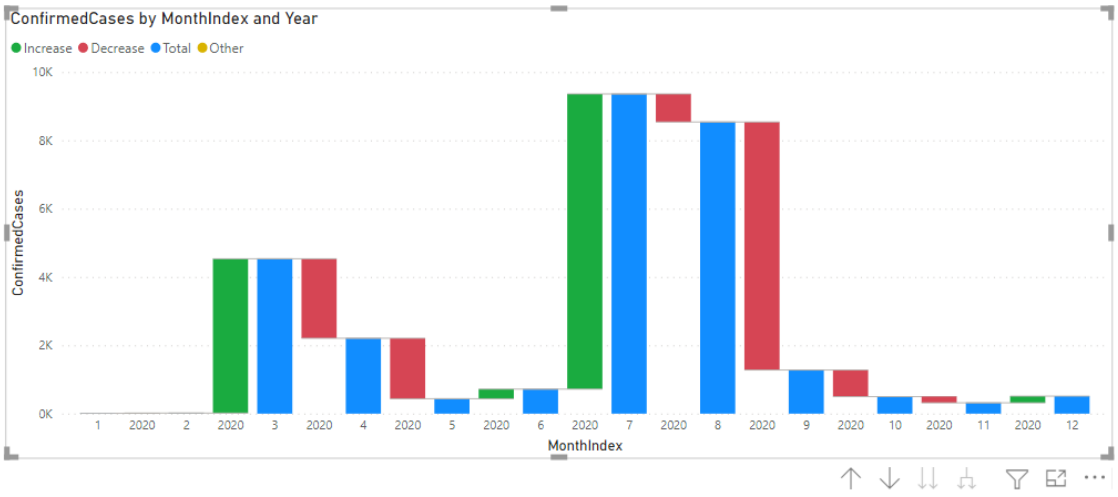
QuarterIndex	Year	CountryName	MonthIndex	ConfirmedCases
1	2020	Australia	1	9.00
1	2020	Australia	2	16.00
1	2020	Australia	3	4,534.00
2	2020	Australia	4	2,207.00
2	2020	Australia	5	436.00
2	2020	Australia	6	718.00
3	2020	Australia	7	9,360.00
3	2020	Australia	8	8,539.00
3	2020	Australia	9	1,277.00
4	2020	Australia	10	499.00
4	2020	Australia	11	317.00
4	2020	Australia	12	513.00
Total				**28,425.00**

This Table show above illustrates the number of Confirmed cases in each month of 2020 in Australia (Roll up). We can change the level time and use different levels of information. For example, below I used the bar graph to get total number of cases in whole of Australia which turns out to be 28986.



The bar graphs also show an interesting pattern in the number of cases. We can clearly see there were two main waves in Australia in 2020 when the number of cases spiked.

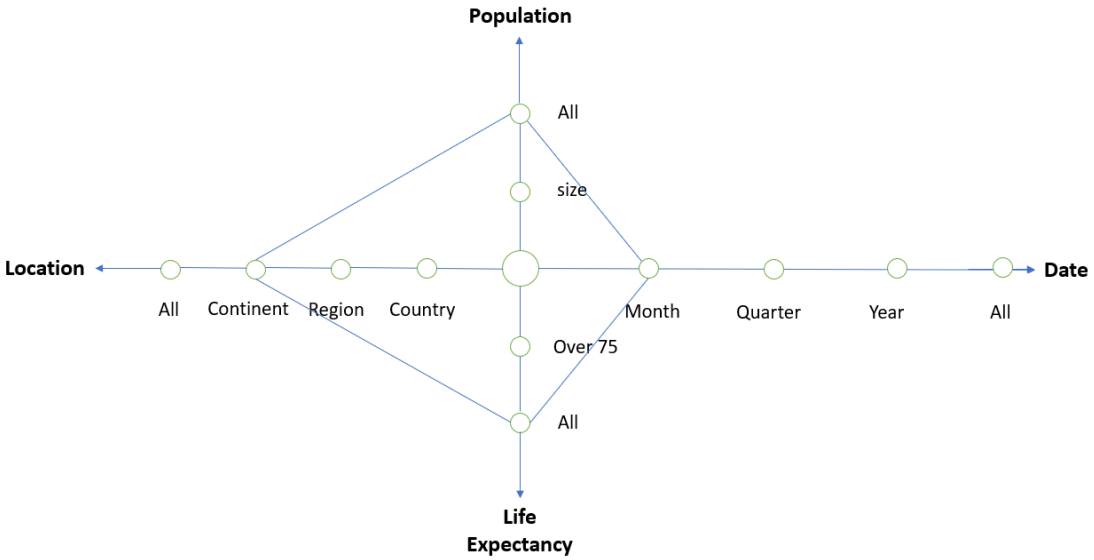
We also used the Slice and Dice Operation in this by selecting Australia as the country that we want to focus on.



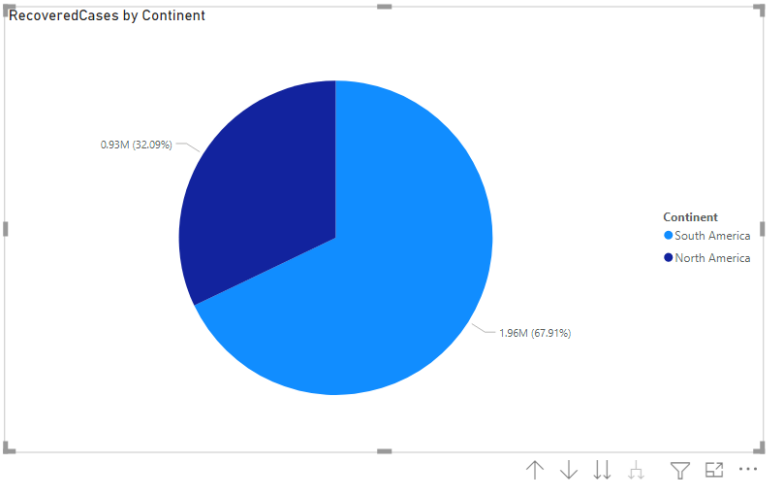
Query 2

In Sept 2020, how many recovered cases are there in the region of the Americas? How many recovered cases in the United States, Canada and Mexico, respectively, in Sep 2020?

Again, we start with getting the StarNet for our query.

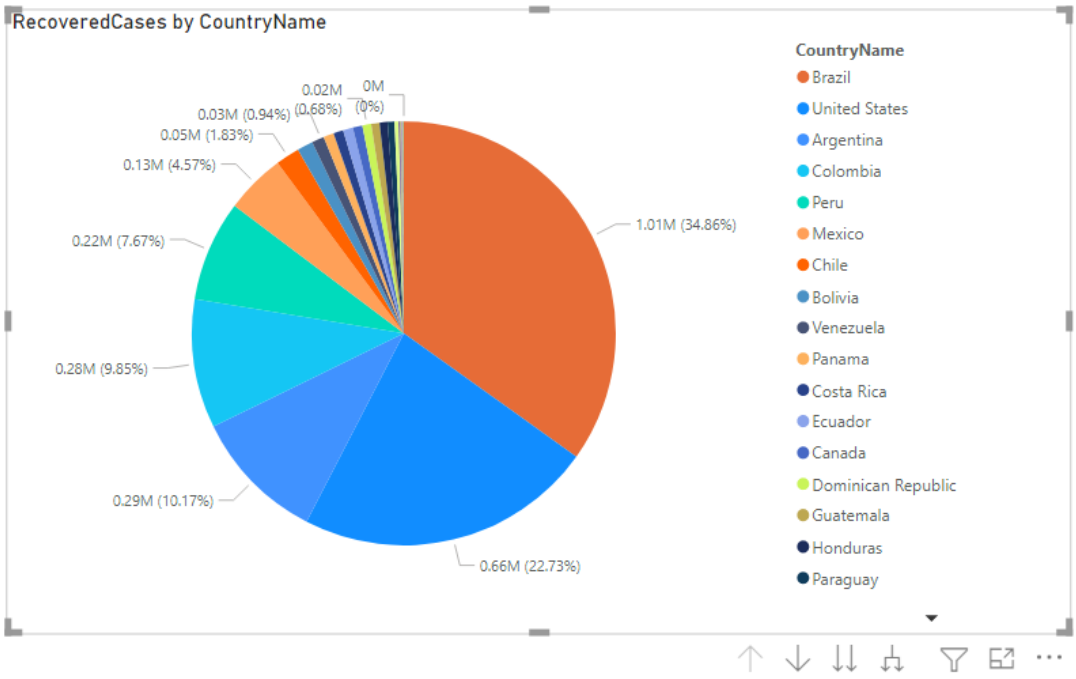


We will use drill down operations here when we go from let's say all to Continent to Country level of information. Similarly, it's roll up if we go from Country level information to Continent level information. We also used Slice and Dice OLAP Operation here as we selected 2020 as our one specific year that we want to focus on.



The pie chart given above demonstrates the number of recovered cases in both South America and North America in 2020. While the table given below shows the number of cases in different countries of these two continents. We can sum both the numbers to get total number of cases in America reason.

CountryName	Continent	MonthIndex	RecoveredCases
Brazil	South America	9	1,006,183.00
United States	North America	9	655,863.00
Argentina	South America	9	293,450.00
Colombia	South America	9	284,193.00
Peru	South America	9	221,468.00
Mexico	North America	9	131,785.00
Chile	South America	9	52,710.00
Bolivia	South America	9	35,121.00
Venezuela	South America	9	27,113.00
Panama	North America	9	22,888.00
Costa Rica	North America	9	21,785.00
Ecuador	South America	9	21,689.00
Canada	North America	9	21,161.00
Dominican Republic	North America	9	19,547.00
Guatemala	North America	9	18,188.00
Honduras	North America	9	17,526.00
Paraguay	South America	9	15,028.00
El Salvador	North America	9	9,521.00

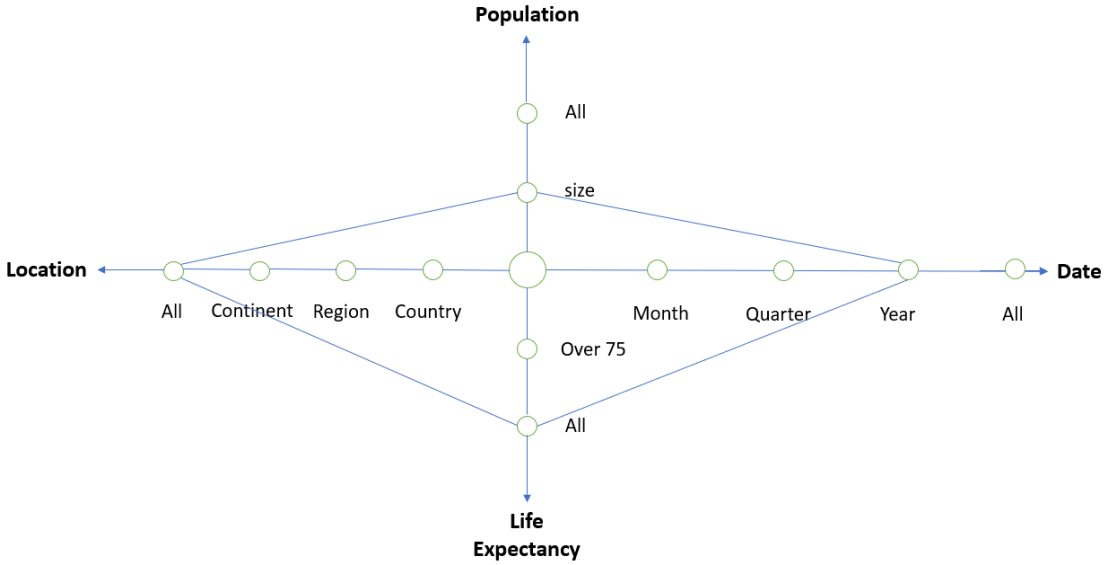


Query 3

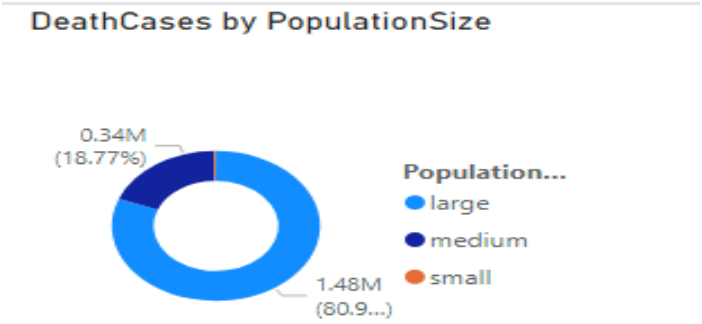
What is the total number of covid deaths worldwide in 2020? What is the total number of covid deaths in large countries, medium countries and small countries, respectively, in 2020?

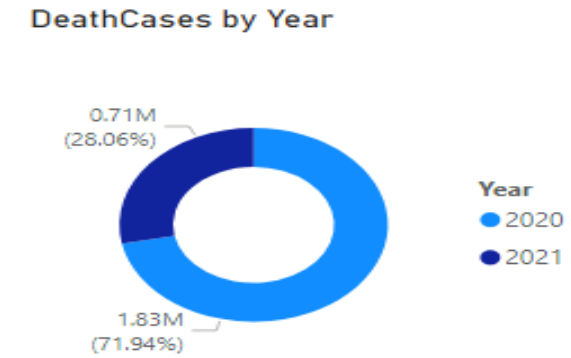
- **Note**: In this project, country size is measured by population. Large countries: population >= 40 million; small countries: population <= 2 million; medium countries: 2 million < population < 40 million.

We again start with StarNet.



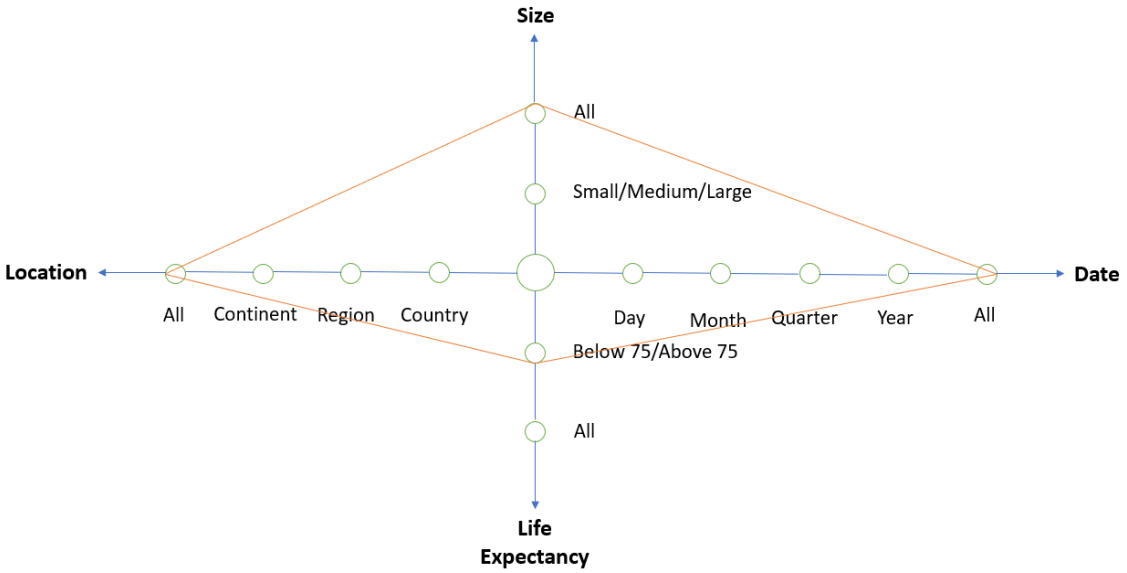
We can use Slice Dice operation to fix the year (time) as the level of information as '2020'. We can then use drill down and roll up Operations on the same Pie Chart to get information for all countries. Note: We are use two different charts here. The first one is most relevant to answer the question while second also answers the second part of the question.



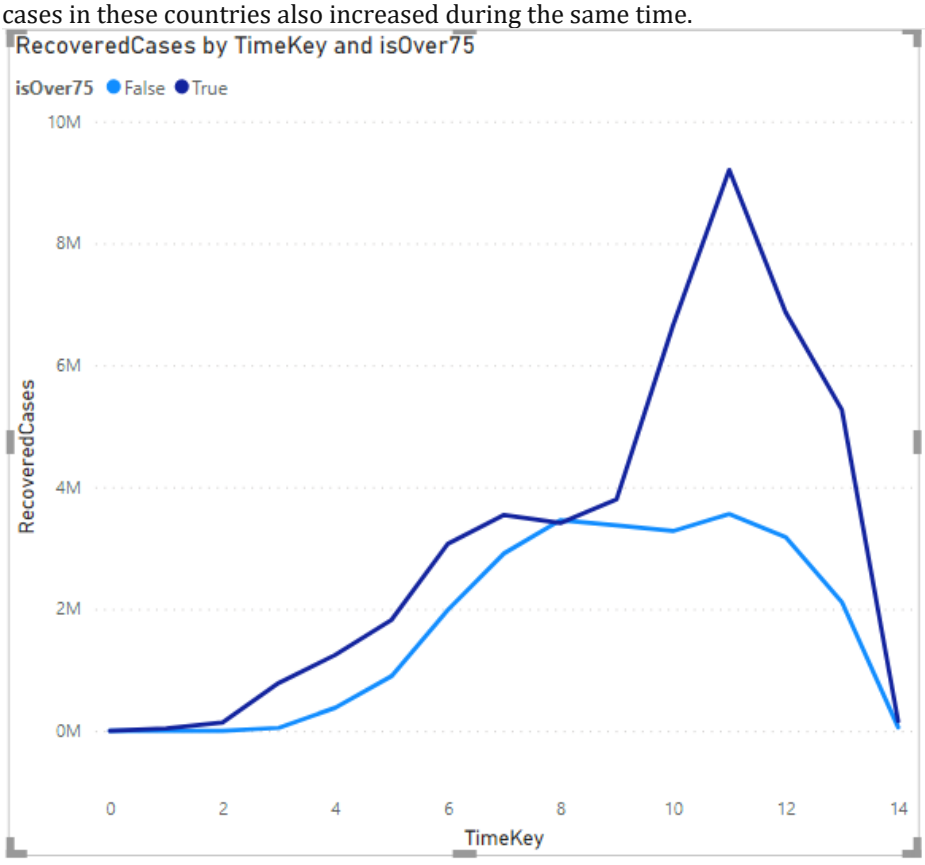


Query 4

Do countries with a life expectancy greater than 75 have a higher recovery rate?



We again started with the StarNet model and implemented the queries. From the line Graph given below, we can see that countries with Life expectancy higher than 75 has a similar kind of wave as countries with life index lower than 75. There's spike in the cases but that doesn't prove anything because the number of cases in these countries also increased during the same time.



The following graph also depicts that countries with life expectancy might not have released the data of recovered cases hence we see a spike in number of cases which isn't seen in recovered cases.