CITS3401 – Project 1 Report

Student Name: Thomas Cleary
Student Number: 21704985

Aim

The aim of this project is to answer the four given business queries:

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?

Part 1 – Design

To create the design for the data warehouse that will be used to answer the four given queries, Kimball's four steps can be followed to create a design template.

1. Identify a business process to model

- In this instance, the 'business process' will be the ongoing worldwide cases, deaths and recoveries of COVID-19.

2. Determine the grain of the business process

- The grain of the business process is, as can be seen in the provided time series data files, daily updates to confirmed cases, deaths and recoveries by country, and where applicable province/city.

  However, it is apparent from the queries the granularity we will actually need is monthly values by country only (whilst also specifying life expectancy and country size for each country).

3. Choose the dimensions that apply to the fact table row

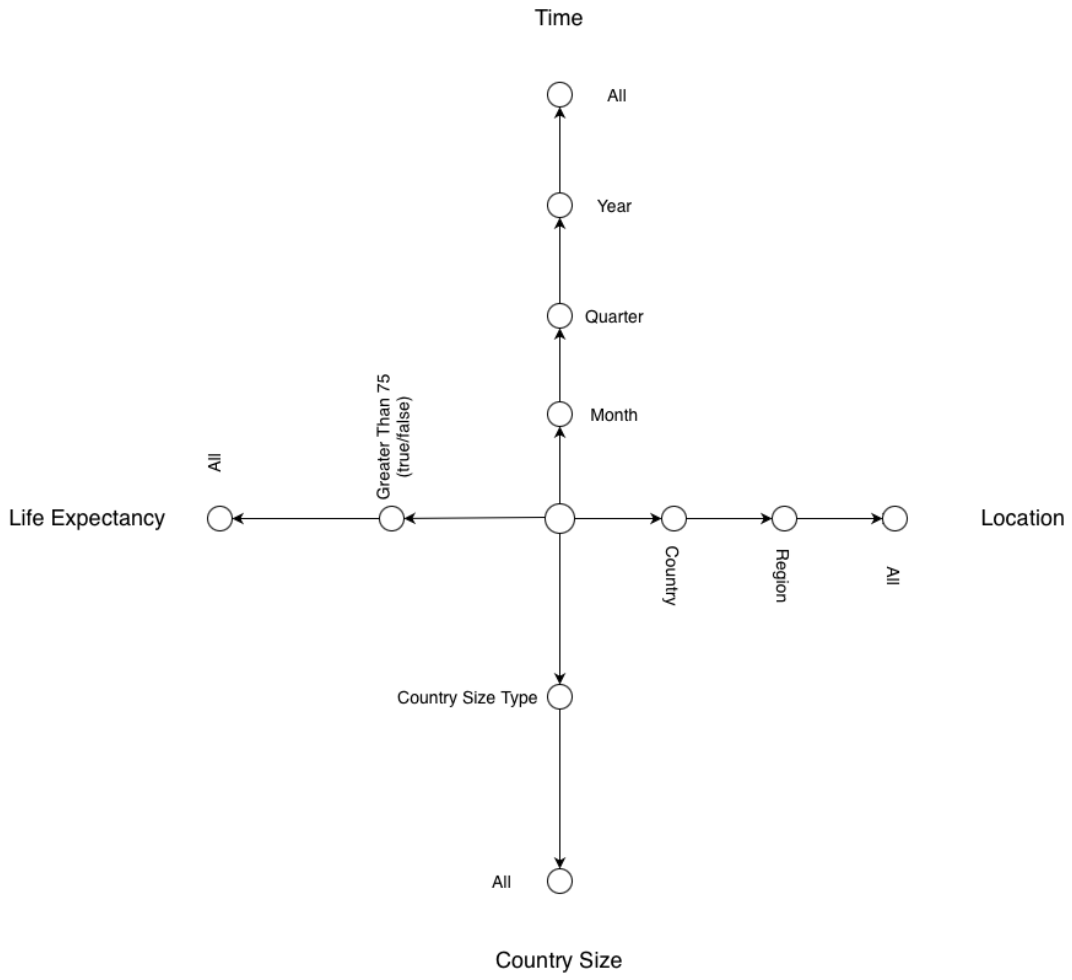
- From the 4 given queries it is clear we will need four dimensions:
  - Time
    - Query 1, 2, 3
  - Location
    - Query 1, 2
  - Country Size
    - Query 3
  - Life Expectancy
    - Query 4

4. Identify the measures that populate the fact table rows

- We can see that three measures will be needed to answer the four business queries:
  - COVID-19 Confirmed Cases
    - Query 1, 4
  - COVID-19 Deaths
    - Query 3
  - COVID-19 Recoveries
    - Query 2, 4

Design – Starnet Model

Now we can create a Starnet model to illustrate the concept hierarchies we will need for each dimension in the data warehouse.



We can now add footprints to this model to determine the granularity we will need for each business query.

Design – Starnet Footprints

For each business query a “footprint” can be drawn over the Starnet model to determine the granularity that will be needed for each dimension in the data warehouse.

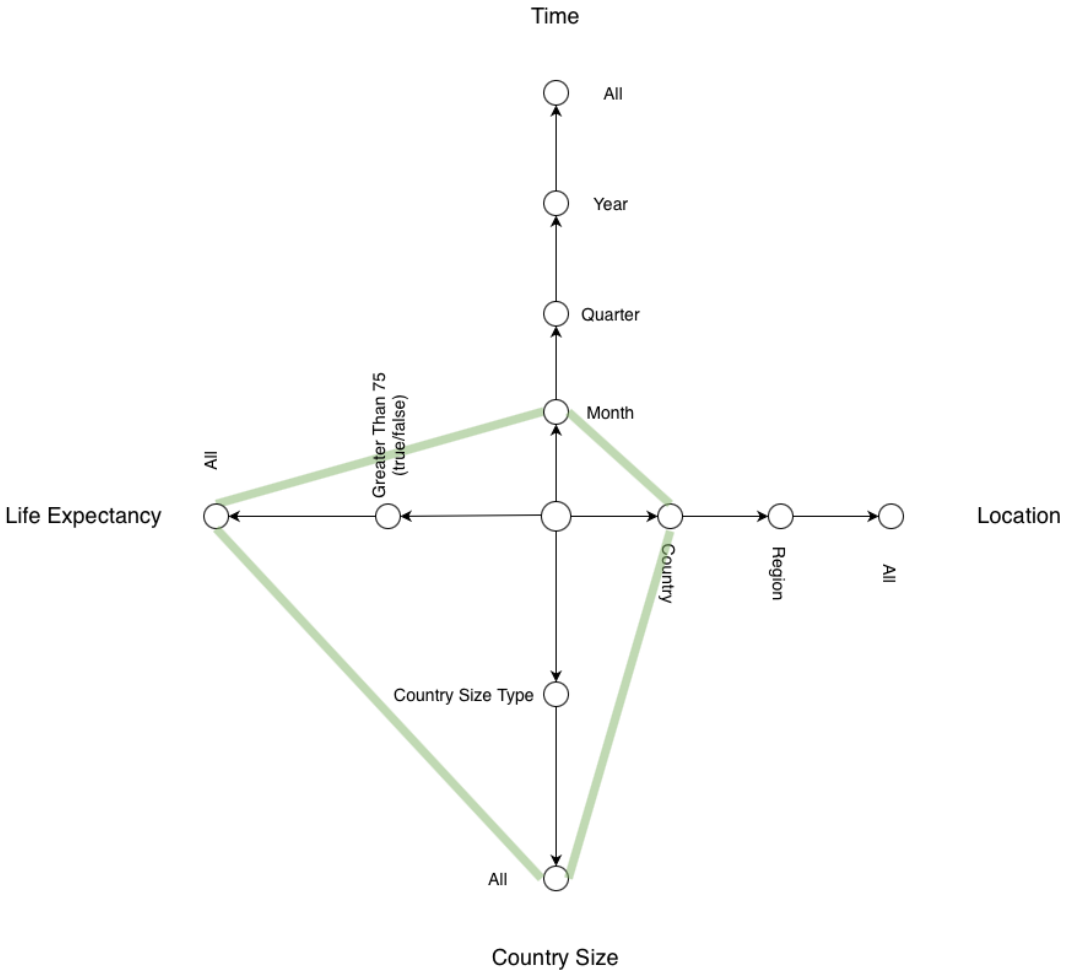
Query 1

- This query requires
    - A specific country (Australia)
    - Each level in the dimensional hierarchy, Time (month, quarter, year)

Query 2

- This query requires
    - Specific countries (United States, Canada, Mexico)
    - Specific regions (North and South America)
    - A specific month (September in 2020)

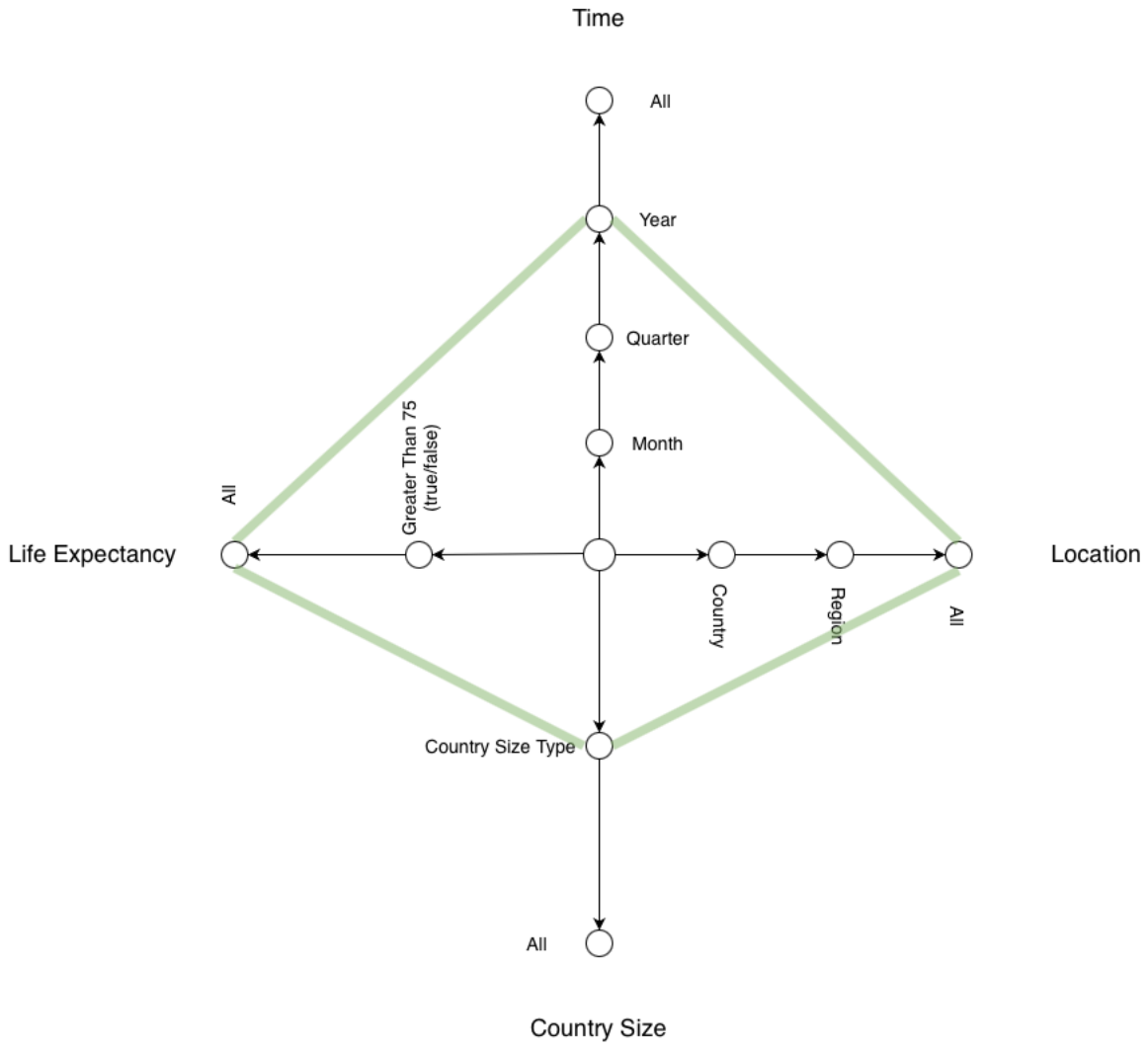
The below footprint illustrates how both query 1 and 2 can be answered with the previous Starnet model.



Query 3

- This query requires
    - A specific year (2020)
    - All countries (data points) to be grouped by their size (small, medium, large)

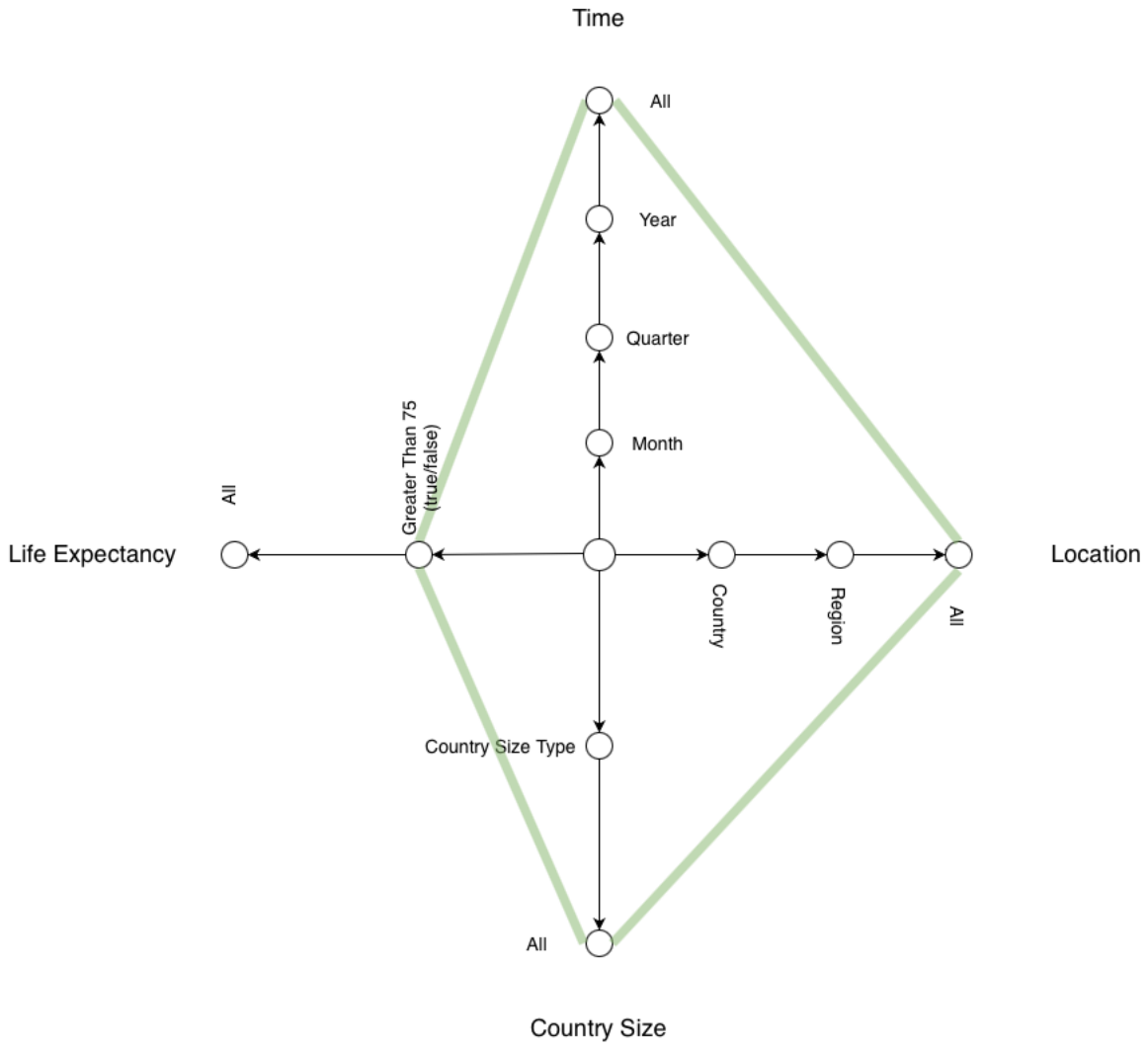
The below footprint illustrates how query 3 can be answered with the previous Starnet model.



Query 4

- This query requires
    - All countries (data) points to be grouped by whether their life expectancy is greater than 75 or not.

The below footprint illustrates how query 3 can be answered with the previous Starnet model.



Part 2 – Implementation

To implement the data warehouse a logical representation of how the data will be stored must be designed, in this case a star schema.

This allows the physical creation of the data warehouse, CovidDW, which can then be used to form a data cube in Visual Studio. This will then allow us to visualise our business queries in PowerBI.

However, to insert the data into the data warehouse we must perform an ETL process. This involves extracting the COVID data from various sources, transforming it to suit our needs and then finally loading it into the data warehouse.

This section will discuss:
- The CovidDW database star schema
- CovidDW database implementation details
- The ETL process on the provided COVID data
- Data cube implementation details

The basic steps I took to implement the data warehouse were:

- Create the database CovidDW with the SQL script “CovidDW.sql” to understand the data I would have to extract, and the form it would need to be in once extracted.

- Extract relevant data from the given .csv and .xlsx data files

- Transform this data to match the types needed in CovidDW (as per CovidDW.sql)

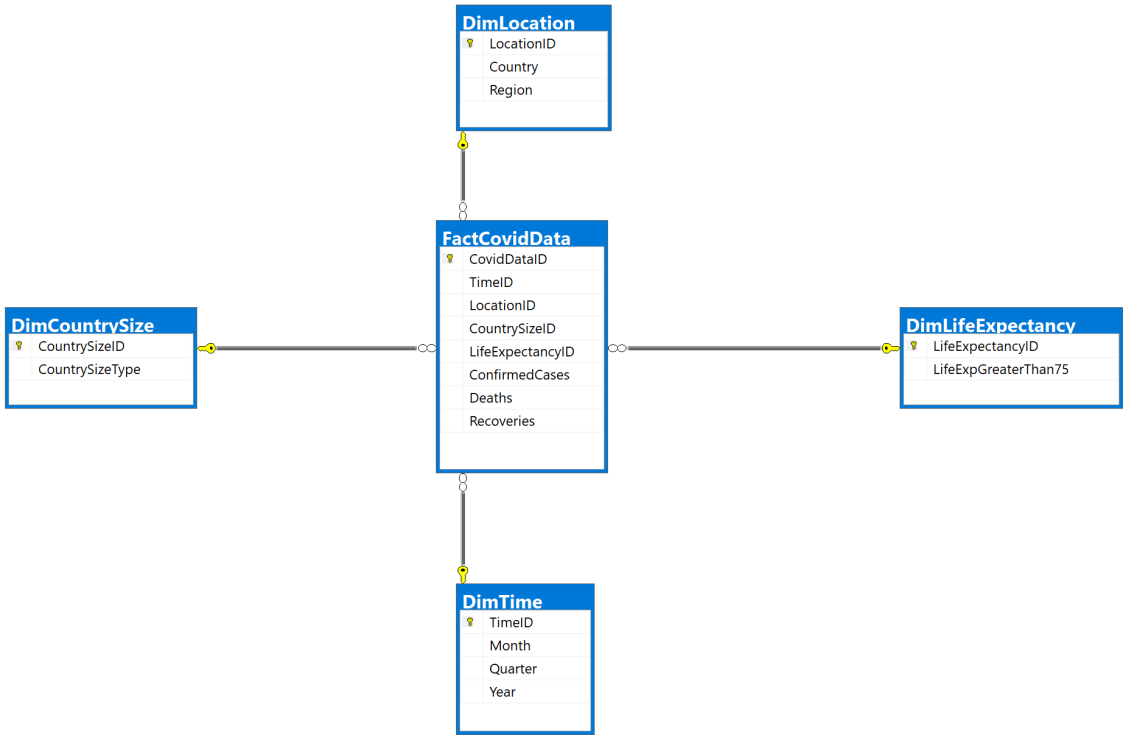
- Write this data to .csv files that can be bulk inserted into CovidDW

- Insert the data into CovidDW via the .csv files

- Create the data cube from CovidDW in Visual Studio

Implementation – Star Schema

The below star schema illustrates the design of the CovidDW database that will be used to create a data cube in Visual Studio and answer the four business queries in PowerBI.



Each table beginning with "Dim" is a dimension table that represents one of the dimensions in the Starnet model shown previously. "FactCovidData" is the data warehouse's fact table. It contains each of the measures (ConfirmedCases, Deaths, Recoveries) as well as foreign keys into each dimension table that's name matches that of the primary key in the respective dimension.

This is clearly a star schema as no dimension contains a foreign key into a further dimension. This reduces the complexity of the database yet increases the redundancy compared to a snowflake schema.

I considered separating the country information out of "DimLocation" into a separate table "Country" and also storing the country size and life expectancy information there. However, decided against it as I believed the reduced complexity would aid the ETL process.

Implementation – Database Creation

First, before beginning the ETL process the script, “createCovidDW.sql”, was written to implement the schema for the data warehouse which is illustrated in the previous star schema. This script outlined the types of each column in each dimension and fact table. As well as defining the relations between the dimensions and fact tables.

An example of one of the table creation statements can be seen below.

```
35  PRINT '';
36  PRINT '*** Creating Table DimLocation';
37  GO
38
39  CREATE TABLE DimLocation
40  (
41      LocationID  INT          PRIMARY KEY,
42      Country     VARCHAR(255)     NOT NULL,
43      Region      VARCHAR(255)     NOT NULL
44  )
45  GO
```

For the dimension tables I decided I would provide the primary key values to make writing the fact table .csv file easier as I would already have information for its foreign keys.

However, the fact table uses an autoincrementing integer as its primary key as I did not need to know the value of its primary key for the creation of any of the .csv files.

I also decided to store Month in the Time dimension as an int instead of varchar like “March” as this made it easier to create correctly ordered visualisations in PowerBI.

Implementation – ETL Process

By inspecting the files provided, I gathered that the "owid" .xlsx file could provide the country names with their respective region, life expectancy and population. The three "time series" .csv files could each provide one of the measures for each country (confirmed cases, deaths, recoveries).

To get this data from the four files into the CovidDW database would require extracting, transforming and then loading the data.

Before extracting the data, a design for a data structure to load the information into was created as seen below.

```
"""
data = {string countryName : {

    # Dimension Table Primary Keys (except DimTime)
    "location_id"     : int
    "country_size_id" : int
    "life_exp_id"     : int

    "region"          : string

    "measures"        : {
        (time_id, month, quarter, year) : [confirmed, deaths, recovered]
    }
}}
"""
```

The plan was to eventually have all the necessary data in a Python dictionary that could be used to write relevant data to individual .csv files required to load the data into each table of the database.

Each country name would be a key into another dictionary that stored that country's data. It would contain:

- Primary keys for the dimension tables
    - DimLocation
    - DimCountrySize
    - DimLifeExpectancy

- The name of the region the country is in

- Another dictionary containing (date : measure) pairs
    - date, would consist of (DimTime primary key, month, date, year) for that particular months measures (confirmed, deaths, recoveries)

Extraction

Data extraction was performed solely with Python and associated libraries within the file "getCovidData.py". There are three major functions within getCovidData.py that do the majority of the data extraction.

- *get_owid_data()*
- *get_time_series_dates()*
- *insert_time_series_data()*

get_owid_data() sets up the data structure (shown previously) with country and region names as well as primary keys for life expectancy and country size from "owid.xlsx". It then calls *get_time_series_dates()* via *init_measures()* to add each (time_id, month, quarter, year) 4-tuple from dates transformed from the "time series" .csv files. Each measure (confirmed, deaths, recovered) is then set to 0 before we extract the actual measures.

Countries like Kosovo that had odd values for life expectancy or population, like 0, were set to less than 75 and small respectively.

Once the data structure has been populated with everything except the values for the measures, *insert_time_series_data*() is called to insert data for each country into each date for all three of the measures.

```
1  ------------------------------------
2  OWID Country Names
3  (Missing in TimeSeries)
4  ------------------------------------
5  Cape Verde
6  Congo
7  Democratic Republic of Congo
8  Hong Kong
9  Myanmar
10 Palestine
11 South Korea
12 Taiwan
13 Timor
14 United States
15 Vatican
16 ------------------------------------
17 TimeSeries Country Names
18 (Missing in OWID)
19 ------------------------------------
20 Burma
21 Cabo Verde
22 Congo (Brazzaville)
23 Congo (Kinshasa)
24 Diamond Princess
25 Holy See
26 Korea, South
27 MS Zaandam
28 Micronesia
29 Taiwan*
30 Timor-Leste
31 US
32 West Bank and Gaza
33 ------------------------------------
```

However, *insert_time_series_data()* relies on the countries extracted from owid.xlsx to match those found in the time series files. Which on further inspection were found not to match up. *create_country_diff_txt()* in "compareCountryNames.py" creates a text file listing all the countries in the OWID data that do not appear in the time series data and vice versa (as seen to the left).

(**NOTE:** All three time series file were checked to make sure they contained the same list of countries with *compare_time_series_countries())*

Clearly there are several countries in both files that do not have a match in the other file.

To fix this issue, I have transformed the data in each time series .csv file before extracting the data.

Transformation

To be able to extract the data from each time series file the country names must match with those found in the OWID .xlsx file. I decided to change the names in the time series files to those found in the OWID data as seen below.

```
31  ------------------------------------------------------------
32  Changes to TimeSeries Files
33  ------------------------------------------------------------
34  Burma              -> Myanmar
35  Cabo Verde         -> Cape Verde
36  Congo (Braz...)    -> Congo
37  Congo (Kins...)    -> Democratic Republic of Congo
38  Holy See           -> Vatican
39  Korea, South       -> South Korea
40  Micronesia         -> Marshall Islands
41  Taiwan*            -> Taiwan
42  Timor-Leste        -> Timor
43  US                 -> United States
44  West Bank and Gaza -> Palestine
45
46  CHANGE China -> Hong Kong where province == Hong Kong
47
48  REMOVE CRUISESHIPS FROM CSV FILES
49  ["Diamond Princess", "MS Zaandam"]
50
51  CHANGE China -> Hong Kong where province == Hong Kong
```

I made the decision to remove the rows where the country was specified as a cruise ship as cruise ships are not countries.

“editTimeSeriesFiles.py” contains a function *changeCountryNames()* that uses regular expressions to find and change each country name in each time series file that is in the above list to its equivalent in the OWID data file.

Now the time series files are in a state that they can be read by *insert_time_series_data()* to add the measures to the data structure. However, there are certain other data transformations that have to be performed whilst we are extracting the data as well.

To obtain the primary key for a countries life expectancy and country size two funtions:

- *get_country_size_id()*
- *get_life_exp_id()*

are used to transform the value of a either the countries population or life expectancy into the respective key in the dimension tables within the CovidDW database.

get_country_size_id() returns either 1, 2, or 3, depending on whether the country is small, medium or large depending on the criteria outlined in query 3.

get_life_exp_id() returns 1 if the countries life expectancy is greater than 75 otherwise it returns 2.

As the dates provided in the time series files are daily we need to use *get_time_series_dates()* to "flatten" the list of dates down from a value for every day (mm/dd/yy) to (time_id, month, quarter, year) 4-tuples eg. (1, 1, 1, 2020) represents time_id = 1, month = January, quarter = first quarter in the year, year = 2020.

With these transformations are data structure can be successfully populated with everything except the actual values of the measures. The more interesting issue now arises that the values for each measure for each country in the time series files are cumulative. If we wish to extract the number of cases for a particular month we will need to remember how many cases happened before this month and subtract this from it when entering it into the data structure.

insert_time_series_data() in getCovidData.py extracts a measure by iterating over each row (except the header) in one of the time series files. For each row we take the country name as our key into the data dictionary, and then proceed to iterate over each data value in its row.

For each row a running total is kept for the current months total and the number of cases before this month. If we get to a data value that is the start of a new month we add the current month total minus the previous amount into the data dictionary by converting the date in the header to a 4-tuple key with *date_to_key()* and updating the relevant measure (confirmed, deaths, recoveries).

In some instances, the value of the measure reduces, implying the country has corrected its value for the measure, giving a negative result for the month. I have decided to leave this in as it would be to hard or even impossible to deduce which previous months values to reduce for these corrections.

An issue arises, specifically found in the time series recovered .csv, where some countries stop reporting values and instead have trailing 0's. If not accounted for *insert_time_series_data()* will add the month with the trailing 0's as having negative the total of previous cases found prior to that month. Therefore, I decided in the case of trailing 0's to in essence stop adding values at the point trailing 0's are found and leave subsequent months values at 0.

Functions are then provided in “writeDimCSVs.py” and “writeFactCSV.py” to write the information from the data dictionary into .csv files which can be bulk inserted into the CovidDW database.

The below image shows the function that writes the .csv file for the fact table.

```
11  def write_fact_covid_data_csv():
12      """ FactCovidData.csv
13      (,DateID,LocationID,CountrySizeID,LifeExpectancyID,ConfirmedCases,Deaths,Recoveries)
14      """
15      MEASURES      = "measures"
16      DATE_ID_IDX   = 0
17      CONFIRMED_IDX = 0
18      DEATHS_IDX    = 1
19      RECOVERED_IDX = 2
20
21      covid_data = gcd.get_covid_data()
22      countries = covid_data.keys()
23
24      lines = []
25
26      for country in countries:
27          country_data = covid_data[country]
28          measure_data = country_data[MEASURES]
29
30          for date_key in measure_data.keys():
31              date_id         = date_key[DATE_ID_IDX]
32              location_id     = country_data["location_id"]
33              country_size_id = country_data["country_size_id"]
34              life_exp_id     = country_data["life_exp_id"]
35              confirmed       = measure_data[date_key][CONFIRMED_IDX]
36              deaths          = measure_data[date_key][DEATHS_IDX]
37              recovered       = measure_data[date_key][RECOVERED_IDX]
38
39              line = ",{},{},{},{},{},{},{}\n".format(
40                  date_id,
41                  location_id,
42                  country_size_id,
43                  life_exp_id,
44                  confirmed,
45                  deaths,
46                  recovered)
47
48              lines.append(line)
49
50      wdc.write_lines(lines, FOLDER + "FactCovidData.csv")
```

Loading

An SQL script was created to load the created .csv files into the CovidDW database. An example of bulk inserting the data for the fact table can be seen below.

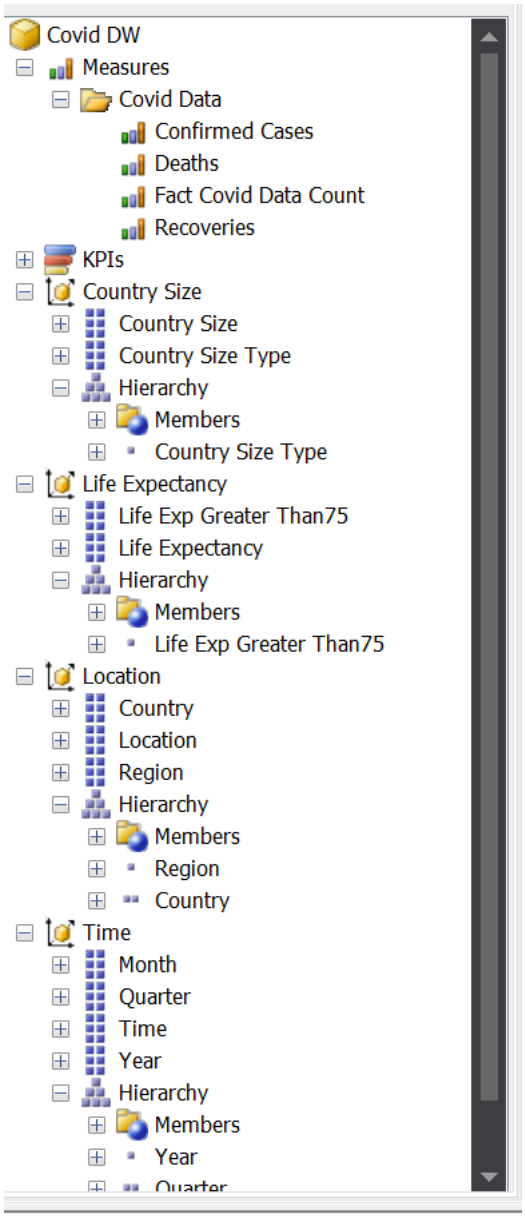
```
60  BULK INSERT [dbo].[FactCovidData] FROM '$(csvFilesPath)FactCovidData.csv'
61  WITH (
62     CHECK_CONSTRAINTS,
63     DATAFILETYPE='char',
64     FIELDTERMINATOR=',',
65     ROWTERMINATOR='\n',
66     TABLOCK
67  )
```

One issue that arose when attempting to insert the data was the ROWTERMINATOR argument being incorrect if the .csv files were created on MacOS. The line ending character encoding on UNIX like systems is slightly different than Windows and required ROWTERMINATOR=‘0x0a’. However, the simpler option was to simply remake the .csv files on Windows and leave ROWTERMINATOR=‘\n’.

Implementation – Data Cube Creation

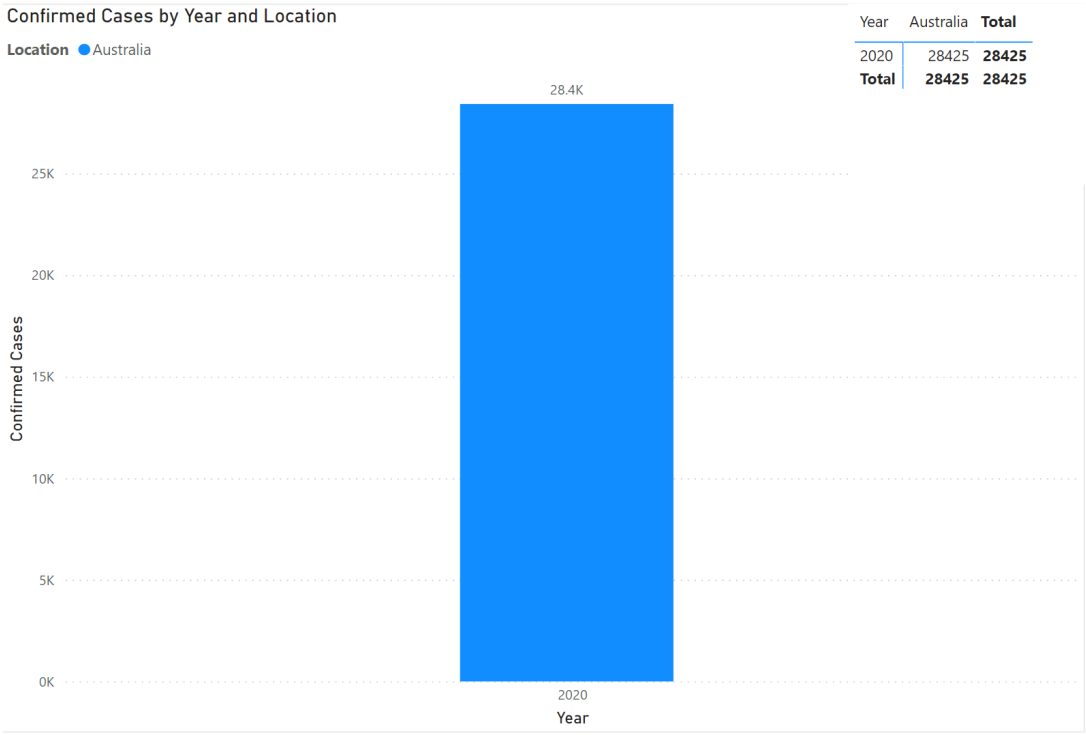
An *Analysis Service Multidimensional and Data Mining Project* was used to create the data cube that will allows us to visualise and answer the business queries.

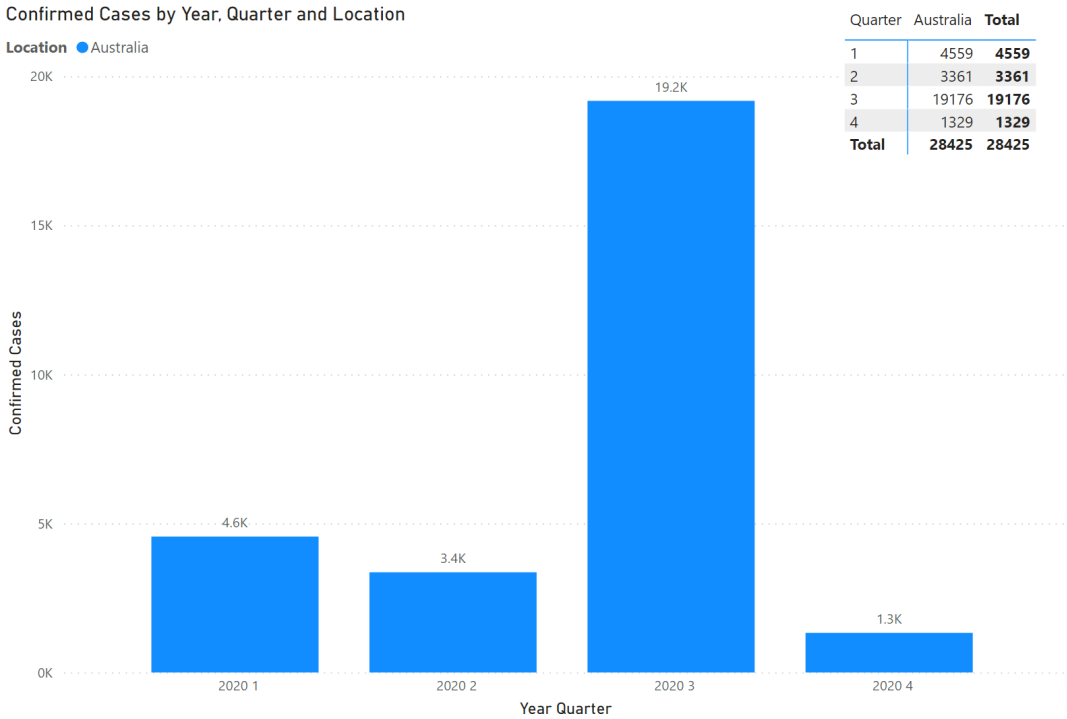
Below is an image of the cubes browser panel that shows the hierarchies designed in the original Starnet model have been successfully implemented into the data cube.

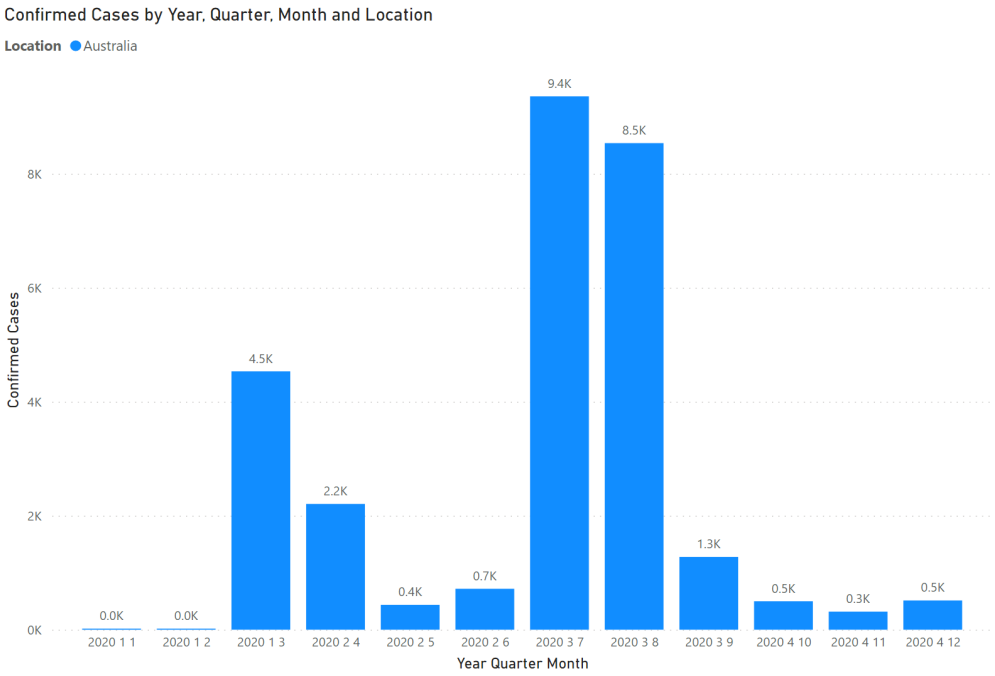


Part 3 – Query Results

Query 1

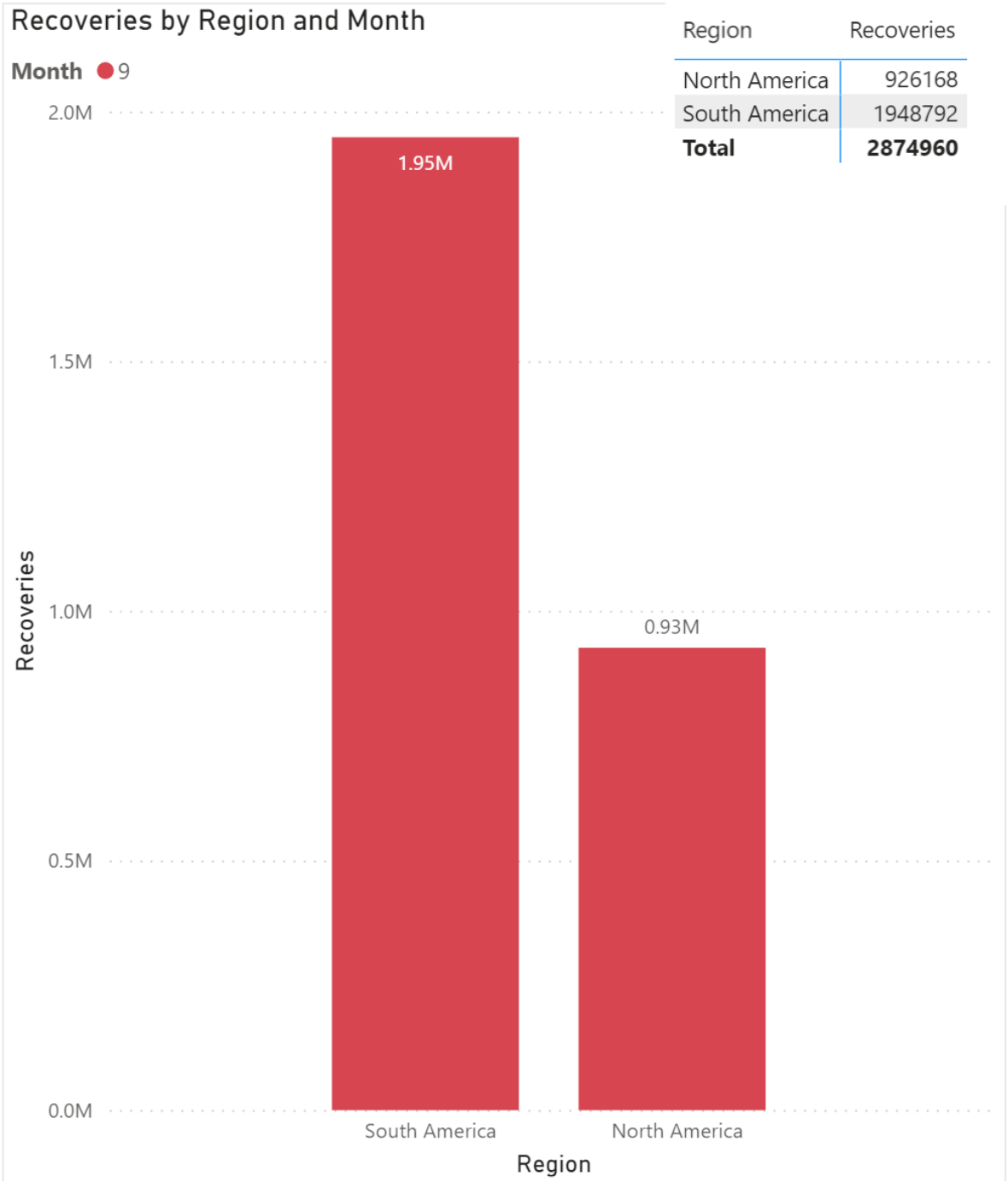




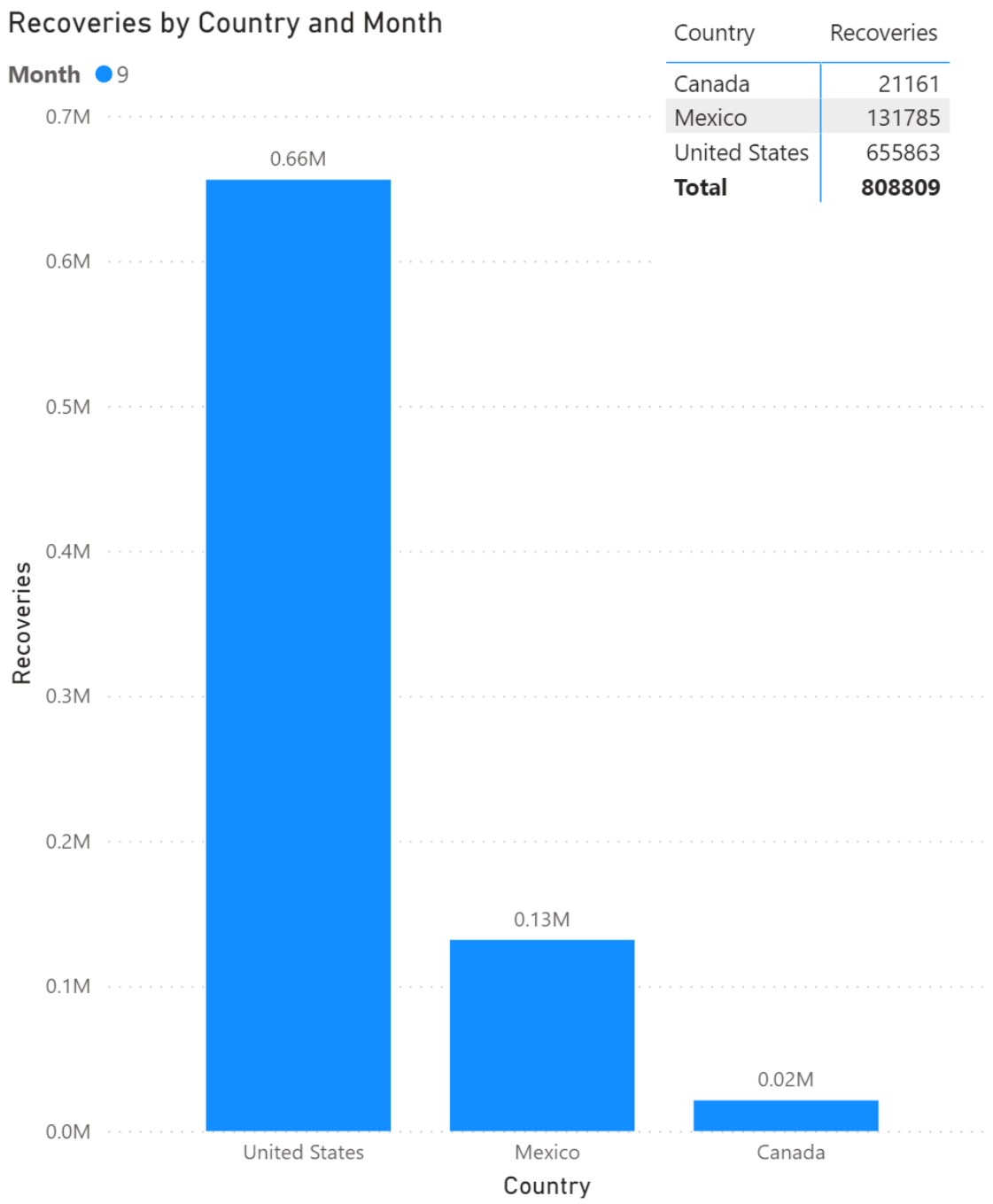


Month	Australia	Total
1	9	**9**
10	499	**499**
11	317	**317**
12	513	**513**
2	16	**16**
3	4534	**4534**
4	2207	**2207**
5	436	**436**
6	718	**718**
7	9360	**9360**
8	8539	**8539**
9	1277	**1277**
Total	**28425**	**28425**

Query 2

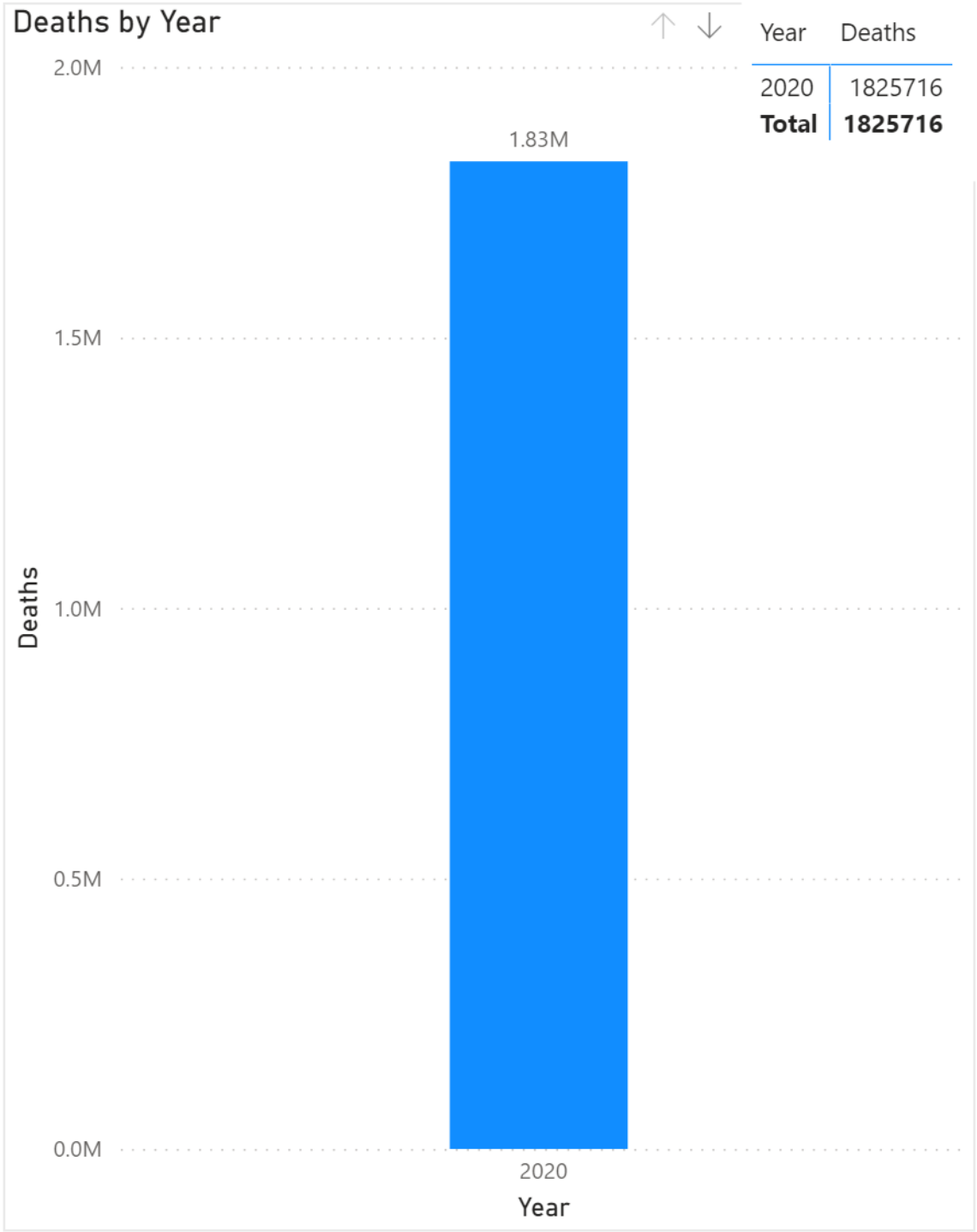


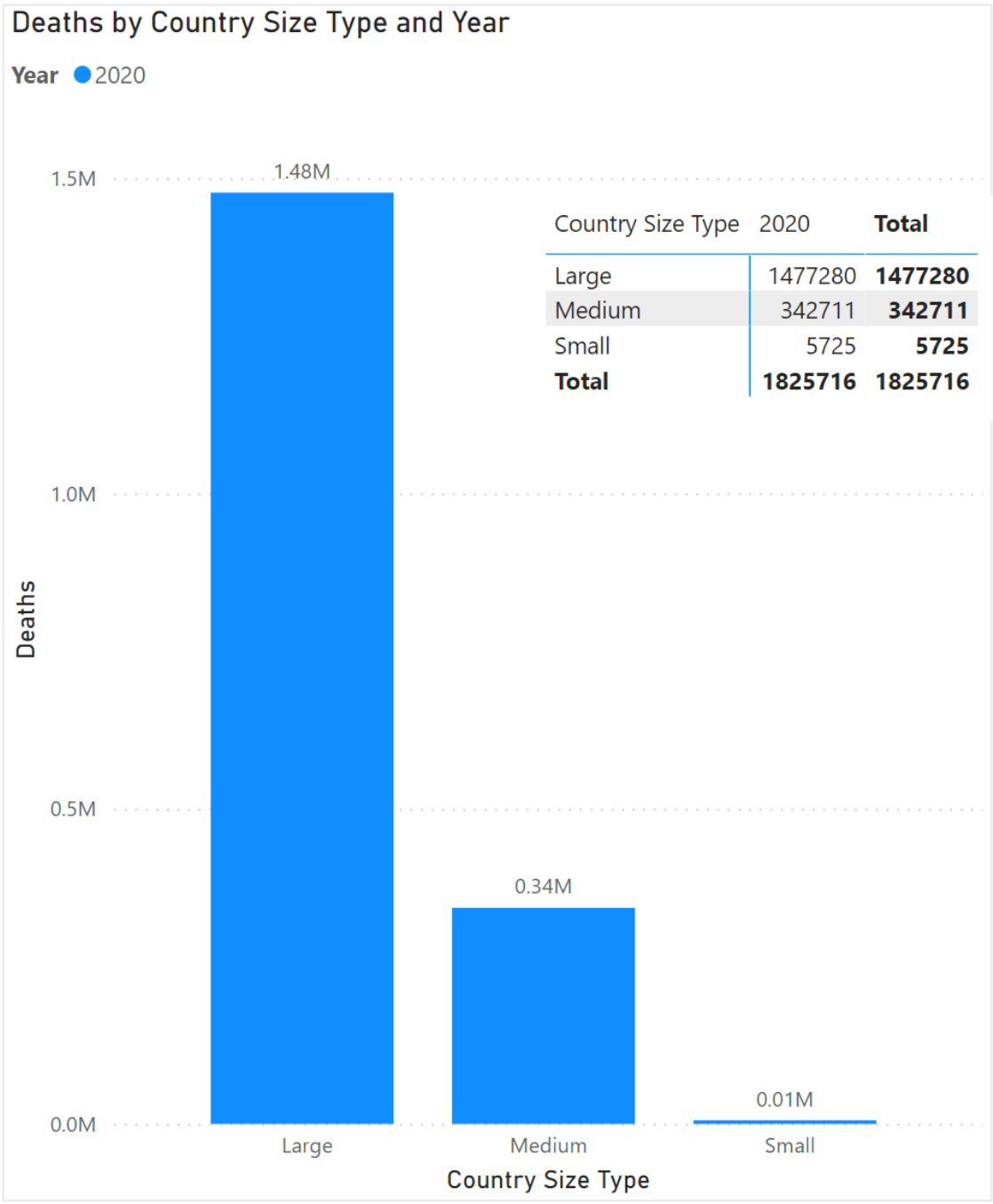
Region	Recoveries
North America	926168
South America	1948792
Total	**2874960**



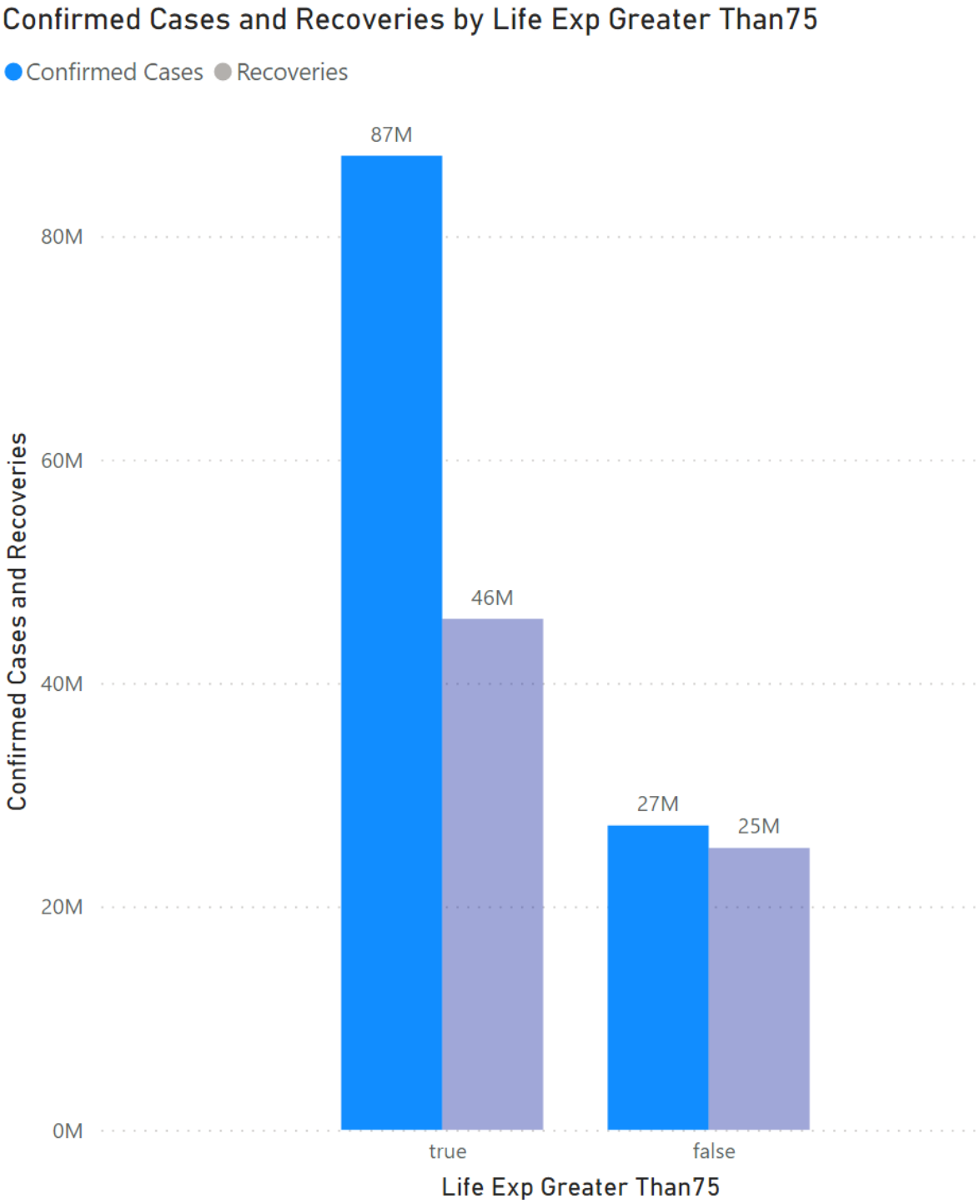
Country	Recoveries
Canada	21161
Mexico	131785
United States	655863
Total	**808809**

Query 3


Deaths by Year
Year
Deaths
2020
1825716
Total
1825716
2.0M
1.5M
1.0M
0.5M
0.0M
1.83M
Deaths
2020
Year


Deaths by Country Size Type and Year
Year 2020
1.5M
1.0M
0.5M
0.0M
Deaths
1.48M
0.34M
0.01M
Large
Medium
Small
Country Size Type
Country Size Type 2020 Total
Large 1477280 1477280
Medium 342711 342711
Small 5725 5725
Total 1825716 1825716

Query 4



Life Exp Greater Than75	Confirmed Cases	Recoveries	Recovery Rate
false	27242950	25237995	0.92640462...
true	87198975	45742448	0.52457552...

Part 4 – OLAP Operations

Query 1

Slice

- The time dimension is sliced by only looking at the year 2020
- The location dimension is sliced to only look at the country, Australia

Drill-Down

- We begin by looking at the confirmed cases for 2020
    - Then drill down to the next level in the Time Dimension, quarter and look at the confirmed cases for each quarter in 2020
        - Then drill down again to the next level in the Time Dimension, month and looks at the confirmed cases for each month in 2020

Query 2

Slice

- The time dimension is sliced by to look at the month September in the year 2020

Dice

- The location dimension is diced to look at the regions North and South America
    - Can now see recovery totals for these regions in Sep 2020

Drill Down

- The location dimension is drilled down to country

Dice

- The location dimension is diced to look the countries Canada, Mexico, United States
    - Can now see recovery totals for these countries in Sep 2020

Query 3

Slice

- The time dimension is sliced to only look at the year 2020
    - Can now see worldwide total for deaths in 2020

Drill-Down

- Introduce the Country Size dimension (drill down by country size)
    - Can now see total deaths for each country size in 2020

Query 4

Drill-Down

- Drill down by Life Expectancy to Life Expectancy Greater Than 75 level